NAVAL POSTGRADUATE SCHOOL

Monterey, California

D784165

THESIS

Network And Database Design in Support of the
Joint Theater Level Simulation

by

Charles Dunn III
September 1988
and
Stanley H. Evans, Jr.
June 1988

Thesis Advisor: Joseph S. Stewart

Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification Unclassified		1b Restrictive Markings	
2a Security Classification Authority		3 Distribution Availability of Report Approved for public release; distribution is unlimited.	
2b Declassification/Downgrading Schedule			
4 Performing Organization Report Number(s)		5 Monitoring Organization Report Number(s)	
6a Name of Performing Organization Naval Postgraduate School	6b Office Symbol *(If Applicable)* 39	7a Name of Monitoring Organization Naval Postgraduate School	
6c Address *(city, state, and ZIP code)* Monterey, CA 93943-5000		7b Address *(city, state, and ZIP code)* Monterey, CA 93943-5000	
8a Name of Funding/Sponsoring Organization	8b Office Symbol *(If Applicable)*	9 Procurement Instrument Identification Number	
8c Address *(city, state, and ZIP code)*		10 Source of Funding Numbers Program Element Number / Project No / Task No / Work Unit Accession No	
11 Title *(Include Security Classification)* Network and Database Design in Support of the Joint Theater Level Simulation			
12 Personal Author(s) Charles Dunn III and Stanley H. Evans Jr.			
13a Type of Report Master's Thesis	13b Time Covered From To	14 Date of Report *(year, month,day)* June1988	15 Page Count 133
16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
17 Cosati Codes Field / Group / Subgroup		18 Subject Terms *(continue on reverse if necessary and identify by block number)* Joint Theater Level Simulation, Postprocessor, ORACLE DBMS, INGRES DBMS	

19 Abstract *(continue on reverse if necessary and identify by block number*

The purpose of this thesis was to determine the feasibility of incorporating a Sun Workstation into a Command and Control station to aid the players in the execution of their roles in the Joint Theater Level Simulation. This entailed reviewing the possibility of eliminating the use of the Postprocessor from the analysis phase of the game play. The Joint Theater Level Simulation is a theater independent computer wargame that models two-sided air, ground and naval combat, utilized for warfare training, joint operational planning and doctrinal analysis. The products of this thesis will interface the Sun Wokstation with the wargame's host computer, the VAX-11, to provide the players the capability to access and analyze game data to improve their decision-making ability. To meet this end, several software products were produced which specifically interface with the VAX-11, Sun Workstation and Ethernet.

0 Distribution/Availability of Abstract [✓ X unclassified/unlimited] ☐ same as report ☐ DTIC users	21 Abstract Security Classification Unclassified	
2a Name of Responsible Individual oseph S. Stewart, II	22b Telephone *(Include Area code)* (408) 646-2493	22c Office Symbol 55St

D FORM 1473, 84 MAR — 83 APR edition may be used until exhausted — security classification of this page

All other editions are obsolete — Unclassified

NETWORK AND DATABASE DESIGN IN SUPPORT OF THE JOINT THEATER LEVEL SIMULATION

by

Charles Dunn III
Captain, United States Army
B.S., University of California, Berkeley, 1981

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

and

MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY
(COMMAND, CONTROL AND COMMUNICATIONS)

from the

NAVAL POSTGRADUATE SCHOOL
SEPTEMBER 1988

NETWORK AND DATABASE DESIGN IN SUPPORT OF THE JOINT THEATER LEVEL SIMULATION

by

Stanley H. Evans, Jr.
Captain, United States Army
B.S., Temple University, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY
(COMMAND, CONTROL AND COMMUNICATIONS)

from the

NAVAL POSTGRADUATE SCHOOL
JUNE 1988

ABSTRACT

The purpose of this thesis was to determine the feasibilty of incorporating a Sun Workstation into a Command and Control station to aid players in the execution of their roles in the Joint Theater Level Simulation. This entailed reviewing the possibility of eliminating the Postprocessor from the analysis phase of the game play. The Joint Theater Level Simulation is a theater independent computer game that models two-sided air, ground and naval combat, utilized for warfare training, joint operational and planning and doctrinal analysis. The products of this thesis will interface the Sun Workstation with the wargame's host computer, the VAX-11, to provide the players the capability to access and analyze game data to improve their decision maaking ability. To meet this end, several software products were produced which specifically interfaced with the VAX-11, Sun Workstation and Ethernet.

DISCLAIMER

Many terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each individual occurrence of a trademark, all registered trademarks appearing in this thesis are listed after the firm holding the trademark:

Digital Equipment Corporation, Maynard, Massachusetts
VAX 11/780 Minicomputer
VMS Operating System

Bell Laboratories, Murray Hill, New Jersey
UNIX Operating System

CDI
Simscript II.5

Sun Microsystems, Inc.
Sun Workstations

Xerox Corporation, Stamford, Connecticut
ETHERNET Local Area Network

ORACLE Corporation, Belmont, California
ORACLE
SQL*PLUS

TABLE OF CONTENTS

I. INTRODUCTION

A. WAR GAMING

Of paramount importance in any combat situation is the ability of a commander to manage his forces and weapons effectively. For the commander to be able to execute his role in the management of these assets, he must be able to continually assess his own situation, and formulate his intentions relative to the most current data available to him at that instant in time.

For many years models, simulations and games have been commonly incorporated as analytical tools to aid those in the "war fighting" business. Not surprisingly, within the last fifteen years there has been a renewed interest in the area of wargaming by the Department of Defense (DOD) as more powerful, reliable and reasonably priced computer systems have been made available [Ref.1]. A role for computers in battle analysis has perhaps finally found its niche within the military wargaming community. At all levels throughout the Department of Defense, emphasis is being placed on wargaming as a very important training and analytical tool [Ref 2]. Up to now the emphasis has been placed on major force and theater level applications. This drive is currently being headed by the Joint Chiefs' of Staff Modern Aids to Planning Program (MAPP), which now provides the commanders in chief with powerful data processing capabilities for evaluating military alternatives in appropriate situations [Ref. 3].

B. PURPOSE

One of those theater-level applications is the Joint Theater Level Simulation (JTLS). Though the game is valid as a simulation of theater level combat activities, in its current configuration, it has a major distraction which prevents maximum exploitation of the training that it offers. Presently, real-time analysis of game-generated data cannot be conducted while the game is being played. Instead, either the game must be temporarily halted and saved for later restart, or the game must be completely terminated [Ref 4]. Then a Postprocessor must be loaded and invoked through a sequence of time consuming steps by one of the controllers or players. This procedure requires the players of JTLS to learn more than just how to play the game. The players are now compelled to also learn how to make use of the Postprocessor in order to complete the analysis and critique phase of the gaming process. The players must therefore, endure both the disruption of stopping the game and the inconvenience of learning how to use another piece of software.

This block of action is extraneous, cumbersome, inefficient and greatly distracts from the play of the game. We propose that the Postprocessor be eliminated altogether from this war-gaming system. In its place an alternative process needs to be developed that will allow the game-generated data to be sent directly to a database management system facilitating immediate use of the data by the player for on-line summary of current situational status.

To preclude the need to continually run the large and cumbersome simulation to test the feasibility of this proposal, a major portion of the research of the thesis will be devoted to the formulation, testing and implementation of the Push, Pull, and Pack programs. The corollary goals will be:

1) to have the data sent through the network by the Push program to be accepted on the Sun Workstation by the Pull program,

2) to populate an ORACLE database with the data accepted by the Pull program via the Pack program,

3) to ensure that consistency exists between the data generated by the game with respect to the data now available in the ORACLE DBMS, and

4) to execute several SQL*C formulated queries to demonstrate that there has been no additional operational restrictions placed on the game play because of the displacement of the Postprocessor.

In order to implement the proposal, we submit that Sun Workstations, which are now available in the Naval Postgraduate School war laboratory, be incorporated into the currently configured local area network. The result of this modification will facilitate the game data being directly placed into the ORACLE DBMS. To complete the feasibility testing, a single Sun Workstation installed with the ORACLE DBMS will be dedicated to the querying of the database to provide this real-time game analysis. The actual number of Sun Workstations will have no imact on the present gaming configuration nor will it impact the development and implementation of the Push program.

The proposed changes, if successfully incorporated, can only enhance the effectiveness of the game. The ability to do real-time/on-line analysis of the game cannot help but promote a greater sense of realism in the simulation game. Subsequently, this will significantly contribute to making the whole realm of simulation gaming a more viable evaluation source of the decision-making process during times of stress and uncertainty. This change will provide a close replication of the demands placed on a decision maker during circumstances that cannot or are not desirable to be recreated in the real world.

C. THESIS OUTLINE

It is the purpose of this thesis to review the current implementation of the JTLS system and to test the feasibility of incorporating a relational database management system directly into the playing phase of the game. In order to accomplish this, the thesis was partitioned into several chapters, each dealing with specific aspects of the current JTLS implementation to include both its software and hardware architectures.

Following this introductory chapter, Chapter 2 is dedicated to the analysis of the actual project. The physical and temporal constraints of this project are enumerated. Detailed explanations to support various design decisions are presented to justify the options selected.

The third chapter includes a very simple description of the JTLS game, Postprocessor and the essential user functions for both. Only major aspects of these functions are discussed to provide the reader an opportunity to understand the peculiarities of the game and the user requirements of the current system.

The fourth chapter consists of an indepth description of the INGRES DBMS, currently used in the query and analysis phase of the game play.

Appropriately, the fifth chapter explores the characteristics of the ORACLE DBMS. This chapter contains several examples of how specific ORACLE features contribute to the flow of game play.

A very basic explanation of a generic local area network (LAN) follows in Chapter 6 with much emphasis on TCP/IP and their role in data transmission across the internet. This chapter is included to familiarize readers with appropriate terms, concepts and general implementation of an abstract local area networking system. For those who have had significant exposure to networking, this chapter provides a very cursory review of pertinent topics in the networking subject area.

The seventh chapter has been devoted to the formulation, elaboration, implementation and analysis of the several software programs which are required to interface with the internet, the Sun Workstation and the ORACLE DBMS. This includes the inception of the various programs, flow charts of the algorithms and finally, a detailed narrative of the actual execution of the programs.

Chapter 8 contains a brief outline of how the Relational Design was developed, coalescing data information from several unrelated technical manuals. The actual Relational Design is available in Appendix J.

The last chapter contains conclusions and recommended areas for further study.

II. APPROACH TO THE PROBLEM

A. INTRODUCTION

The question of why the Postprocessor was initially created must be answered in order to appreciate its role in the current configuration of this simulation. Discussion with Rolands and Associates Inc., the current contractors of JTLS, reveals that the CALTECH Jet Propulsion Laboratory (JPL), the original contractors of the war game, wanted JTLS to be a distributed war-gaming system that would work as fast and efficiently as possible [Ref. 5]. With the complete separation of a database from the war game, the contention for processor time on a single processor system was no longer a point of concern. The VAX machine became a dedicated machine for the execution of the game. Therefore, the subsequent use of the Postprocessor, after the game had been stopped, had no impact on the execution speed of the game. The tradeoff between a fast game and the inconvenience of periodically stopping the game while waiting for the Postprocessor to read the many files to prepare the INGRES database was soon reassessed. The six to eight hours that were required for the Postprocessor to transfer the gaming data for an eighteen to twenty-four hour play period became a point for discussion, and a remedy was sought to reduce this wasted time [Ref. 6].

B. PROBLEM CONSTRAINTS

1. Application

The basic characteristics of JTLS will be considered invariable for the purpose of this thesis. The actual execution of the war game will have no impact on the development and performance of the Push, Pull and Pack programs.

2. Language

The selection of SIMSCRIPT II.5 as the programming language of JTLS will not be changed. Aside from the obvious reason of having to transpose the entire algorithm into another language, a more difficult task is the selection of a suitable alternative simulation language. SIMSCRIPT II.5 is a programming language especially developed to be used for this type of application [Ref. 5]. Its close resemblance to FORTRAN makes it a language of choice for many of the original programmers of JTLS who were already quite familiar with FORTRAN.

SIMSCRIPT II.5's intimate compatibility with VMS allows the exploitation of the peculiarities of this operating system, resulting in fast and efficient execution of the game code, of the system calls and of all other lower level subroutines that other wise may not have been possible [Ref. 5]. As is the claim of so many other languages which are available today, SIMSCRIPT II.5 is extremely readable, requiring minimum documentation. More importantly, it is very flexible, with provisions for access to many features of the VMS operating system which are not so readily available to other programming languages. Additionally. SIMSCRIPT II.5 is a very efficient higher-level language with an extremely high execution rate which is so essential in a sequentially read simulation program attempting to retain as close to a real-time scenario as possible.

3. Programs

The Push program is being developed to assist in testing the feasibility of reducing, and possibly totally eliminating, the use of the Postprocessor from the playing sequence in JTLS. This program accommodates the testing phase of this proposal by simulating the data handling activities of the Combat Events Program (CEP). Through the implementation of the Push program, the need to repeatedly run JTLS as a data source to populate the ORACLE database for demonstration purposes is alleviated. After the termination of an initial run of JTLS, the Push program will use the saved files that were generated as its source of data for later use. The Push program must be able to extract the data from these several files.

Using the SEND and RECEIVE interprocess communication primitives available with TCP/IP, the data is sent via the Ethernet from the VAX to a Sun Workstation. There another set of programs, called Pull and Pack, were written to accept, parse, and finally populate the appropriate tables located in the ORACLE database. A hard fast constraint which was strictly enforced is that the transported data cannot be modified from it inception at game start through the time it is presented as information to an ORACLE based query.

The reasons for the selection of FORTRAN as the language of choice for the Push, Pull, and Pack programs are several. FORTRAN has proven to be a very readable language and very easy to document, which is indeed a major advantage over such available languages as C. SIMSCRIPT II.5 was originally considered as the language for the programs to maintain uniformity and total compatibility with the gaming code, and to reduce maintenance inconveniences. However, it was not chosen because of the local lack of availability of appropriate programming manuals and the scarcity of any experienced point of consultation when unable to surmount a SIMSCRIPT II.5 oriented programming hurdle. No other languages are currently supported by the VAX system in the Naval Postgraduate School war laboratory.

4. Database Management System

The U.S. Army decided to adopt INGRES as the Army-wide standard database management system. In response to this decision, JPL developed the Postprocessor to work in conjunction with the INGRES DBMS. Therefore, the INGRES DBMS is the relational database that the Postprocessor populates with data from the game files. INGRES DBMS was the preferred database for this research project. However, due to the limitation of funds, the war laboratory was not able to procure a copy of the INGRES DBMS for our use.

Instead, the ORACLE DBMS was made available. It too, is a relational DBMS with many of the same features offered by the INGRES DBMS. Because the differences between the two DBMS' are minimal, the ORACLE translations of the INGRES queries were anticipated to be completed with minor effort.

5. Hardware

The selection of the computer system cannot be changed be cause JTLS was designed and developed to be run exclusively on the VAX machine. Attempts to transport this war game onto another computer will entail a large effort in modifying the original algorithm to accommodate the software and hardware specifications of the other system.

C. ANALYSIS

1. Procedure

Three different aspects of the proposed system were thoroughly analyzed. First, the ability of the Push, Pull, and Pack programs to correctly send and receive the time-sequenced data were extensively tested. Second, the correctness and consistency of the data processed by each program were ascertained by simply comparing the data with those in the game generated data file. Finally, to aid in the development of the database tables and to ensure the correctness of these tables, a comprehensive and thoroughly documented Semantic Data Model and Relational Design were constructed.

2. Database Population

Currently JTLS is operated as a stand-alone simulation with no dedicated DBMS. This precludes a rigid and controlled analysis of the game situations from being performed as it is being played. For such analysis to take place, the game must be halted and the Postprocessor invoked. Following the invocation of the Postprocessor, the players must wait until all the data have been read from the 69 various game generated files. The time it takes to transfer the data is related to the amount of data that is available on the files. Experience has shown that for a game that has been played for approximately eighteen hours, the transfer time is between six to eight hours. Following this inordinate amount of waiting to populate the INGRES database, the players are finally able to query their data source for situational summaries and, pertinent battle and logistics reports.

The thrust of the thesis, as already outlined, are manifold. Initially the creation of the Push, Pull, and Pack programs must be completed. These programs will lend credence to the proposal that a database system on an internet can being run on a separate computer. The final result is the elimination of the Postprocessor and consequently, the availability of an on-line, real-time querying capability. Next, a complete Relational Design for the JTLS database must be developed. The model must be used to construct the needed tables to be populated by the game. The Pack program will then access those tables and populate them with the required data. The populated database is made accessible to the players as the game is in progress. The Push program will simulate the game in progress with the time-stamped data being placed into the Ethernet at the specified times.

3. <u>Data Consistency</u>

Expected data consistency is perhaps the most essential part of this entire project. Without the proper transfer of data, the entire feasibility test fails regardless of whatever else has been accomplished. Although the Postprocessor is cumbersome and time intensive to use, information is nevertheless being generated. Should the Push program fail to transfer consistent data, then less information is made available to the game players. No deviation from what the game generates can be tolerated.

Several runs were be conducted to help generate a consistent set of data for analytical purposes. It is assumed that the network is reliable [Ref. 9].

III. JOINT THEATER LEVEL SIMULATION

A. INTRODUCTION

For the purposes of this thesis a complete description of the user interfaces or the application program characteristics will not be included. Literature to that effect is already available in such publications as Postprocessor User Guide [Ref. 4], and JTLS Executive Overview [Ref. 5]. Instead, only those minimal features which contribute to the reader's appreciation of the complexity of a simulation game will be presented.

B. JOINT THEATER LEVEL SIMULATION OVERVIEW

The JTLS is a computer-assisted wargaming system that models two-sided air, ground and naval combat. The system runs on Digital Equipment Corporation VAX minicomputers, to include the 11/780, 11/785, and 8600 [Ref. 5].

The noteworthy points that set JTLS apart from almost all other wargaming models become evident after noting the enormous effort placed in the design and development of an efficient and effective total system. These considerations are outlined below:

1) The primary software language, SIMSCRIPT II.5, was de signed for creating simulations.

2) The user-machine interaction permits inputs and outputs to be available at independent dummy terminals.

3) A message-handling system and screen menuing capability is provided to the user.

4) An expandable memory capacity allows increased database requirements to be accommodated [Ref 4].

5) The design facilitates future product improvements.

6) Configuration management procedures provide for ongoing visibility and control of software and documentation.

7) A rather important user feature is the capability to run the game faster or slower than real-time. During large scale battle problems, a great deal of time may be spent searching for the opponent's forces,preparing to receive logistics, preparing positions, etc. If

the problem were to run at real-time, the participants may spend many hours waiting for the enemy to be detected. By allowing the game to run faster than real-time it is possible to shorten the detection phase of the problem. Once the actual conflict begins, the warfare simulation can be run at real-time speed or slower.

8) Possibly the most impressive feature of JTLS is that the game runs as close to real-time as is possible in a program that is sequentially executed. This means that the user has the same amount of time to act or react to tactical situations as he would in real conflicts. It is this feature that makes JTLS such an effective training tool. This powerful feature will be carried further by the added ability to analyze the game in real-time. The ability to query the database to determine what is happening at any particular moment enhances the sense of realism. Of course due to the "fog of battle", not all information should be readily available to the players. However, this lack of information or misinformation will be deliberately programmed into the game algorithm, further simulating the "heat of battle".

1. Current Configuration

The current JTLS system is a sequentially executed, user interactive, event-driven simulation. In a future version of JTLS, it has the potential of becoming a distributed war-gaming system. However for now, the system software is executed on a single, central stand-alone VAX computer. All the executive functions are thus performed on that single computer.

In this configuration the JTLS executive functions are divided into two main groups. These function are the user interfaces and the execution of the simulation game. The user interface function is performed through the Model Interface Program (MIP). To activate each user's MIP, the player must go through a login procedure at the start of game play. These terminals are the only means by which a player will make his input available to the game.

JTLS can currently support a maximum of 28 workstations/terminals and 10 graphic screens [Ref. 5]. Two of the terminals are used by the technical coordinator whose responsibilities include:

1) start-up of the graphic processor,
2) control of the Postprocessor program,
3) start-up of the CEP, and

4) saving the output from the execution of the CEP.

At least one terminal will be made available to the game controller. In this capacity, the individual will

1) be able to modify the data parameters in the initial database,
2) be able to modify the logistics for any player in the game, and
3) be able to control the movement of the units in the game.

To manage the simulation of the commander's role and decision making process in actual combat, the Blue Commander and the Red Commander will each be assigned one terminal. From his respective terminal each commander has the ability to observe the graphics screen, hold staff meetings, request reports and dictate guidance. The remaining 23 terminals may be used by supplementary game controllers (if required), additional Commanders, Air players, Intelligence players or Logistics players [Ref. 5]. In addition to the textual information available on the terminal screens, each of the available graphic screens may be assigned to one or more players of the same side. The minimum configuration for game execution consists of at least four workstations. There is one station dedicated to the Red Commander, one station to the Blue Commander, one station to the game controller, and one station to the technical coordinator [Ref. 5].

2. The Model Interface Program

JTLS receives user commands through the MIP. The MIP is an interactive program called by each of the players located at an active terminal. A player MIP provides continuous interaction between the CEP and the player.

The MIP provides the users with the following capabilities:

1) entering orders;
2) processing orders;
3) communicating between players and controllers;
4) communicating between players and the combat simulation;

5) accessing and using support information;
6) saving directives in archive files;
7) analyzing Postprocessor data;
8) controlling graphics output; and
9) stopping or temporarily halting the game.

The number of stations and MIPs needed is a user variable, and is dependent upon the exercise or the system application. Through the MIP each player is able to transmit his decisions, in the form of orders, to the CEP.

The CEP is the warfare-simulation model around which JTLS is developed. The modules included in the CEP simulate the movement and interaction of land, air and sea forces for the two-sided combat.

C. POSTPROCESSOR

The JTLS Postprocessor is a tool that is used to generate information for wargame analysis. The primary software tool used by the Postprocessor is the Interactive Graphics and Retrieval System (INGRES), a commercial relational database management system produced by Relational Technology, Inc. INGRES is augmented by a menu system for ease of use and a separate user-controller data recording and assimilation mechanism. The Postprocessor can be used during a game pause or after game termination.

1. Initiation During a Game Pause

A game pause occurs when the game sequence reaches a CHECK POINT command inserted by a controller throughout the play of the game. Within a few seconds after the recognition of the command, each player's monitor will display the word PENDING on the far right-hand side of the first text line visible on the screen. Each player enters READY to cause his MIP to save important data into a file.

The player that logged on the JTLS system as PLAYER01 is designated the Primary Controller for the JTLS exercise. The key responsibility of the Primary Controller, beyond the responsibilities shared by the other Controllers, is the decision to initiate the Postprocessor during the game pause.

When the Primary Controller invokes the Postprocessor, the data preparation phase commences. During this preparation phase all the output produced since the last time the Postprocessor was used or since the beginning of the game (whichever is appropriate) is transferred from the game generated files into the data tables available in the INGRES DBMS [Ref. 8]. Next, INGRES examines the data it has assimilated and modifies the internal storage of the data to allow for quicker response to a player's queries.

2. Initiation After Gaming Session

To run the Postprocessor in "stand-alone' mode (outside the context of the gaming session) the player must know the name of the Postprocessor database that will be used for analysis. Once the player has initiated the JTLS Postprocessor, he is ready to retrieve and analyze data from a particular Postprocessor database.

The Postprocessor classifies queries in a hierarchical fashion. Beginning at the Entry Menu, the player decides which of the four broad classes of information is of interest. These four classes are combat systems, logistics, air assets, and targets [Ref. 4]. After making the first choice, the player makes further choices, narrowing the query to a specific topic. The player directs the search by entering the number that is displayed on the menu next to his choice.

If the query requires the Postprocessor to produce a report, the player will be prompted for a report title immediately after he selects that option. The report, with the security classification and title, will be displayed on the terminal screen.

IV. INGRES (Interactive Graphics and Retrieval System)

A. INTRODUCTION

INGRES is a database management system well suited for a wide variety of applications. For example, application programs written in C may access INGRES databases through a SEQUEL interface [Ref. 8]. Skilled database users can meet their information management needs by utilizing QUEL, an interactive non-procedural query language. For those using INGRES in conjunction with the JTLS game, a menu-driven set of queries is provided.

B. GENERAL DESCRIPTION

INGRES is one of several DBMS' that does not maintain any functional distinction between attributes and domains [Ref. 8]. These two terms are often used interchangeably in literature dealing with INGRES. However, in this thesis the terminology will be restricted to the word attribute. Also of particular interest is that this lack of an explicit domain does not preclude any theoretical implications arising from the definition of relations from the cartesian product of a set of value domains [Ref. 8].

1. Database Constraints

The only global assertion which applies to the entire data base is the distinction between the database owner (DBA) and other users. There is a system relation known as the USER'S file which contains the information specifying which databases can be opened and by whom.

The creation and destruction of databases are tightly coupled to the VMS operating system. As a result, INGRES enjoys the flexibility, power and security of the VMS file management system. Also, INGRES is written in C, and therefore requires a C compiler which is available on the VAX-11 located in the war laboratory.

2. Operations

In JTLS the game controller is given the responsibility of the Database Administrator. The game controller is privy to a selection of commands unavailable to the normal user. Two such commands relevant to the state of the database are:

> CREATDB - Establishes a new, initially empty database with a given name. The user who issues the CREATDB command is the owner of the database.
>
> DESTROYDB - Destroy a database, whether it is empty or not.

A relation is defined as a subset of the cartesian product of N sets of attribute values [Ref. 8]. It is generally assumed that the user's perception of relation is an entity over which functions and/or predicates can be evaluated. It is also possible to visualize a relation as a table in which the tuples are always removed when relations are updated.

There are a number of operations available in QUEL which relate to the relation and are extensively called by the Postprocessor:

> CREATE - Creates a new relation with a given name. The user issuing the CREATE command is designated as the relation owner. The owner of each relation in the game is the game itself as it creates each needed file for data input.
>
> COPY - Appends the data in a VMS file to an existing relation generated by the game.
>
> DESTROY - Deletes a named relation from the database.
>
> INDEX - Creates secondary indices on existing relations.
>
> MODIFY - Defines the storage structure for a relation by specifying storage organization and keys. This information is available in Appendix J.
>
> SAVE - Changes the default relation expiration date.

These relation operations may be issued only by the owner of a relation.

3. Views

Views are defined as a set of dynamically derived relations [Ref. 9]. A view structure is essentially a relation structure which has its operations restricted.

This function is extensively used in JTLS to insure that game players cannot access tables belonging to their opponents.

Views are defined from relations in the database by the use of the DEFINE command. This command is generated by the Postprocessor, which also supplies the parameters for the command. As with all QUEL commands the game controller has the option of overriding the DEFINE command as he may deem appropriate. View definition can be specified as a subset of the values in the base relation by means of a qualification statement identical to those used in retrieval commands. No other form of view manipulation is possible. All forms of retrieval on the view are fully supported.

Although views are directly derived from relations, they cannot be manipulated as relations can be. Updates are supported if and only if it can be guaranteed that the result of updating the view is identical to that of updating the corresponding real relation.

4. Tuples

A tuple is an instance of a relation. It is implicitly defined when the relation is created. Keys are defined at the relation level and only for purpose of defining storage structures.

There is a wide variety of operations in QUEL for manipulating tuples. Since the query language for INGRES is based upon relational calculus, tuples are selected from a relation which is represented by a tuple-variable. A tuple-variable is defined by use of the RANGE statement. Once a tuple-variable is defined, the definition remains in effect until it is redefined or the game controller ends the QUEL session. Operations for manipulating tuples are completely controlled by the Postprocessor and include:

APPEND - Adds a tuple or tuples to an existing relation.

DELETE - Removes one or more tuples from a relation.

REPLACE - Modifies one or more attribute value in one or more tuples of a relation.

RETRIEVE - Retrieves a subset of the tuples from a relation. Each of these operations can include an optional qualification involving tuple-variables. These qualifications select a subset of the tuples in a relation represented by a tuple-variable.

Tuples may be ordered, but only if the storage structure for the relation has key ordering. When new relations are formed from the retrieval of tuples or a subset of the attributes of the tuples, duplicates are always removed. Tuples retrieved for display do not have duplicates removed unless the UNIQUE keyword is specified in the RETRIEVE statement.

5. Attributes

The names and characteristics of attributes are defined when the relation which contains them is defined. Attributes in conjunction with tuple-variables can be used in QUEL qualifications statements. These attributes are combined in qualification statements with boolean algebra and relational operators as well as implicit existential quantification. The power of the quantification statements is extended by the inclusion of a large library of computational and aggregation functions.

C. SYSTEM ARCHITECTURE

1. Operational Aspects

INGRES provides a high level of access control with the DEFINE PERMIT command. This command is issued by the Postprocessor to restrict the access by opponent players to the relations and/or attributes of a relation. This command is very flexible because it also allows restriction of the type of operation which may be performed like retrieval, update, etc., as well as time-date access constraints. Data dependent access control is supported since the PERMIT command allows a qualification to be specified which restricts access to a subset of a relation tuple.

Though as INGRES is presently employed with JTLS, concurrency is not an issue, INGRES can support concurrent update at the discretion of the game controller. It uses a preclaim algorithm to avoid deadlock [Ref 8]. The locking granularity is at the relation level but has the capability of working at the page level.

2. Benefits of a Relational Database

As a fully relational database, INGRES clearly provides the following conveniences and facilities that otherwise would not be available:

SIMPLICITY - INGRES interfaces are based upon highly usable extensions to the relational calculus. As a result, a small uniform set of operations provides a wide range of selective power.

UNIFORMITY - As the basis for INGRES operations, the relational calculus exhibits closure [Ref. 8]. Closure is a desirable property which simplifies user interaction with the database.

DATA INDEPENDENCE - As a fully relational database, INGRES provides a high degree of data independence.

OPTIMIZATION - As implemented by the Postprocessor, information about the storage structure themselves allow the storage structures to be optimized [Ref. 8]. This optimization leads to faster response times for commonly used retrieval specifications.

BASIS FOR HIGH LEVEL INTERFACES - The data independence, simplicity and uniformity of INGRES data representation and operation make high level interfaces possible and practical.

SECURITY - Security seems to be a distinct advantage to relational databases since the access control rules for DEFINE and DEFINE PERMIT use the same techniques as other operations like retrieval and update.

V. ORACLE DATABASE MANAGEMENT SYSTEM

A. INTRODUCTION:

Relational Software Incorporated (RSI) began development of ORACLE database in 1977 and demonstrated a prototype relational system in 1978. The first copy of the ORACLE Database Management System (DBMS) was delivered for public use in June 1979. RSI claims that ORACLE is a DBMS especially designed for those personnel with minimal computer experience.

RSI also promotes ORACLE as especially useful in an environment in which the DBMS must interact with several different computer hardware systems and different operating systems. Several versions of the software have been developed specifically to exploit the peculiarities and idiosyncrasies of specific machines and operating systems. A version of ORACLE exclusively designed for use on the Sun Workstation, and developed to interact and exploit system services on the UNIX is now available for installation in the War Laboratory. These characteristics enhance the speed and efficiency that queries can be processed by ORACLE.

B. IMPLEMENTAION

These several features have proven to be significant factors in the currently favorable interest expressed by the proponents of JTLS.

1. General Description

The hardware requirements of the ORACLE DBMS are very similar to those of INGRES. A memory allocation of at least 4 megabyte is required. The disk space of 14,000 1K blocks for ORACLE DBMS distribution and task images are also a necessity. The database uses at least 4500 1K blocks for the user database files and before image files. The Sun

UNIX operating system release 3.2 or later is needed to run ORACLE on the Sun Workstation.

The ORACLE DBMS architecture takes advantage of several Sun UNIX features that allow an ORACLE system to share both executable code and data among several programs. These features are outlined below:

1) System Global Area - The SGA is a shared memory region that permits multiple applications or users to access the same data. All ORACLE programs must have access to the common data structures that the SGA contains:

   a) locks,

   b) queues,

   c) transaction recovery information,

   d) process control information,

   e) system configuration information,

   f) before image buffers,and

   g) database buffers.

   Each active ORACLE system has one SGA which contains the locations for its database files and before image files.

2) Shadow process - In the two task architecture of ORACLE implementation, each application task starts a corresponding shadow task. The shadow task constitutes the ORACLE kernel, and is the only foreground process that has access to the SGA.

3) Interprocess Communication - Interprocess communication is the ability to transfer either control or data information from one process to another. Under Sun UNIX, a variety of two- task drivers may be used for this purpose.

2. <u>Data Structure</u>

The ORACLE database system is composed of a database, and its relations, views, tuples, records and domains. These database components are interrelated in the following manner. A relational database consists of relations of possibly many different types. A relation consists of tuples of the identical type. A view is derived from one or more relations by means of a projection and the subsequent use of the qualification operations. A tuple consists of item values of possibly different types. A domain is defined in terms of a user defined data type.

An ORACLE database consists of a set of relation definitions which are stored in system-maintained relations. It also consists of a set of stored tuples for each relation and a set of views. A view is simply a set of virtual relations defined on base relations [Ref.9]. A database is named by assigning the user created name to the file when the database is initially created.

A relation is defined as a two dimensional table of data items. This allows the visualization of a relation as a table of rows and columns. These relations within a database are assigned unique names at the time of definition. In ORACLE duplicate tuples are permitted, although the system will enforce a uniqueness requirement for a primary key if it is specified in the definition.

ORACLE supports a uniqueness requirement for any attribute the user wishes to use as a primary key. Primary keys must be indexed. In addition, the system will reject the NULL values in inserted or updated tuples. The first attribute defined for a relation must be assigned a value when a tuple is inserted.

3. Set Operations

Qualification in ORACLE is calculus oriented [Ref.10]. Qualification results may be thought of as a relation populated with qualified rows. Qualification can be used with retrievals, updates and deletion.

Set operations are not directly supported by ORACLE. Instead, they are accomplished using combinations of other SQL operators. JOIN, in ORACLE, is handled by means a of restriction in the selection criteria subject to the following constraints:

1) reflexive JOINS require using a different table name for each table reference to the base relation, and

2) up to 255 relations can be joined, and

3) items used for specifying JOINs need not be indexed.

4. SQL

The queries for the INGRES DBMS were written in QUEL. Since the queries used by the players were entirely menu driven, a great deal of interaction was required between the INGRES-based application programs and the VMS operating system. This same situation is repeated with the use of the ORACLE DBMS. The queries will again be totally menu driven, requiring the ORACLE application programs to interface with the UNIX operating system. In ORACLE this will be made relatively simple by using the SQL*C language provided.

SQL*C is a self-contained language used for data definition, selection, query, update and interfacing with other compatible environments. It is also used to define relations and attributes, to insert, modify or delete tuples, and to retrieve relations or projections on relations. Finally, it can be used to take advantages of system routines already available in UNIX or used to create needed subroutines to achieve a desired effect. The SQL*C commands are relatively straightforward using a great number of mnemonics and English keywords. For example, selection is based on relational calculus with the criteria specified in a WHERE-clause. The language is user-driven and is normally intended for ad hoc query and update. To accomplish this SQL*C can be used on a stand-alone basis, but more usefully it can be embedded in user-written programs to facilitate the construction of menus and program interfaces.

5. Security Features

As with the INGRES DBMS, views are again extensively used in ORACLE to accommodate the JTLS game play. Views are used to limit player access to specific columns of a table or to selected tuples. Views also provide a means for restricting types of access such as read or write and by whom. In ORACLE, views are defined as dynamic virtual tables comprised of a selected portion of the database. Since the views are derived by selecting qualified tuples from the base relations, the keys are inherited from the base relations.

In addition to the VIEW feature, the Sun UNIX provides additional security assets. These features include file ownership, group accounts and the ability to have a program change its user-id upon execution.

Security is also enhanced by the two-task architecture of the ORACLE DBMS implementation. A division of work and address space exists between the user program and the ORACLE program. This allows an enhanced security scheme since all database access is achieved through the shadow process and special authorizations on the ORACLE program.

VI. THE OPEN SYSTEM INTERNET MODEL

A. INTRODUCTION

Within Department of Defense (DOD) the internet protocol most commonly used is IP [Ref. 3]. For this reason the U.S. Army has expressed much interest in utilizing this network component in its distributed war games. The current configuration of JTLS will need to be modified to include the capacity to incorporate internet communications. This proposal is currently under consideration by the responsible proponents of JTLS.

For the sake of standardization, the generic network system is designed as a series of layers with each layer performing a specific function called a protocol. Through this hierarchy of protocol layers, it is made much easier for one computer system to establish a communication link with another computer system, and pass the desired data correctly across the network. Also this system of hierarchy precludes any of the higher levels being concerned about any of the low level functions required. To provide communication services, each layer must exchange the required signals with its peer layer across the network.

Stratifying the network system allows each layer to view its set of lower layers as providing the services needed to complete its role in the networking hierarchy. The means by which the lower layer accomplishes this task is of no concern to the upper layer. Of particular interest in this thesis are the roles the IP and TCP layers play in the network hierarchy.

B. INTERNET PROTOCOL

1. DESCRIPTION

The Internet Protocol (IP) is designed to provide the necessary functions to deliver a package of bits (datagram) from one host to another [Ref. 11]. The IP has a number of features which enable the protocol to send datagrams across connected networks. The first of these features allow the IP datagrams to be fragmented into smaller packets. This is extremely useful when the intervening networks do not permit packets as large as the created intact datagram to cross. These fragments can then be reassembled at their destination using information contained in the IP header [Ref. 11]. Since it is expected that all data generated by the game will be sent over a single network, this feature will not be elaborated.

For most networking tasks, the minimum underlying raw data transfer services provided by the Data Link Layer is too limited. Thus, the second major feature of the IP is to provide the needed power to transmit data through the internet at the Network layer. Basically, the job of the IP is to interconnect one or more packet handling networks into an internet. The IP provides its services to various upper layers by assisting the delivery of these data packets through the internet. The IP is limited to the basic functions required for delivering a datagram through an internet. Each IP datagram crossing an internet is an independent entity, unrelated to any other datagram [Ref. 11]. The host's IP layer provides services to the Transport layer and relies on services from the Data Link layer. The IP layer takes data sent by a Transport layer and uses the services of the Data Link layer to forward the data to the IP layer of the destination host.

The IP does not promise a reliable service. Hosts receiving IP datagrams will discard the datagrams when insufficient resources are available for processing, and will not detect datagrams lost or discarded by the Data Link layer.

The IP insulates the upper layers from any network specific characteristics. To accomplish this an additional service provided by the IP includes selectable levels of transmission behavior involving such characteristics as precedence, reliability, delay and throughput. The IP also allows data labeling which is needed in secure environments to associate security information with data.

2. Implementation

Transmission begins when a protocol of an upper layer passes data to the IP for delivery. The IP packages the data as an internet datagram and passes it to the Data Link protocol layer for transmission across the local net. The IP sends the datagram through the network directly to the host.

The IP does not only provide services to the upper level protocols. It requires support from the lower levels, including a transparent data transfer between hosts with a single subnetwork as well as error reporting. Datagrams may not necessarily arrive in the same order they were supplied to the subnetwork layer, nor is data guaranteed to arrive error free. The lower level provides reports to the IP indicating errors from the subnetwork and lower layers, as feasible. The specific error requirements of the subnetwork layer are dependent on the individual subnetwork. Ethernet, the DECNET software in use in the war laboratory, does not generally report errors except, for example, when a particular packet needs to be discarded because of 16 consecutive collisions [Ref. 12]. Since the IP datagram delivery is not considered infallible, how an IP module will react to information from a lower layer about the disposition of a particular packet is largely unspecified.

C. TRANSMISSION CONTROL PROTOCOL

1. Description

Generally TCP provides services at the Transport Layer and IP provides services at the Network Layer. The Transport layer is designed to provide a machine with end-to-end subnet independent connections and transaction services. The lower layers of the International Standardization Organization (ISO) model are concerned with the transmission, framing and routing of packets between machines. The Transport layer, however, has the task of providing reliable and efficient end-to-end transmission services between processes rather than simply between machines. All four levels - physical, data link, network and transport - work together to provide a complete transport service. Providing the robust and transparent communications upon which upper levels of protocols may then be built.

TCP is designed to operate over a wide variety of networks and to provide virtual circuit service with orderly, reliable transmission of the user data. The virtual circuit concept is implemented by associating a series of packets with one another. The goal of this association is to provide a service by which applications can talk with one another just as though they had a physical point-to-point link [Ref. 11].

2. Implementation

TCP supports a wide range of upper level protocols which need to send data to their peers on the other host. TCP does not attempt to impose any structure on the data sent by a given upper level protocol. It simply treats the upper level protocol data as though it were a continuous stream, thereby leaving all notions of message structure in the hands of the upper level protocols themselves. TCP does however, attempt to segment the stream into discreet units so it can be sent and received in individual packets. Each of these packets is called a segment.

To maintain a reliable host-to-host connection for the purpose of transferring data, the TCP functions include establishing internet connections, transferring data and ensuring adequate flow control. A major function of TCP is to provide data connections between pairs of upper level protocols. Before any data transfer can occur, a connection has to be made between the two hosts. TCP does this through the use of a three-way handshake. Port numbers are assigned to each end of the connections to identify the logical channels to which the data should be sent at each host. TCP is also responsible for breaking the connection once the application layer is finished. This activity is referred to as connection management. Connection management can be broken down into three phases:

1) connection establishment,
2) connection maintenance, and
3) connection termination.

Connections are endowed with certain properties that apply for the lifetime of the connection, including security and precedence levels. These properties are specified by the upper level protocols at connection openings. TCP provides a means for a upper level protocol to actively initiate a connection to another upper level protocol uniquely named with a socket. A socket is actually the concatenation of an IP address with the application's port number [Ref. 11]. A connection is defined by the combination of the two participants' socket numbers.

Once a connection has been established, TCP will maintain it as long as both parties are interested in keeping it active. Connections which are established but which are not actively sending user data do not generate any packets. This is not a problem, but it is interesting that TCP does not specify a mechanism for detecting the loss of a connection partner when no data are being exchanged. But since for some applications such information is of use, some TCP implementations use a trick to accomplish the detection. They send a datagram with no data and an incorrect sequence number. TCP specifies that the recipient must respond with a datagram indicating the correct sequence number. If no response is received, the probing TCP may be able to decide that its peer has disappeared.

Established connections can be terminated in either of two ways:

1) Graceful close - Both upper level protocols close their side of a duplex connection, either simultaneously or sequentially, when data transfer is complete.

2) Abort - When one upper level protocol unilaterally forces closure of the connection, TCP does not coordinate connection termination.

Flow control mechanism permits a receiving TCP to govern the amount of data dispatched by a sending TCP. The mechanism is based on a window which defines a contiguous range of acceptable sequence numbered data. As data are accepted, TCP slides the window upward in the sequence number space. The current window is specified in every segment and enables the peer TCPs to maintain up-to-date information.

VII. PROGRAMS

A. INTRODUCTION.

The files, code, code description, and variable descriptions for programs are in Appendices A through L. The flowcharts are in Appendix L. The Appendices are provided to compliment the following discussion, and to support the actual implementation of the feasibility test.

B. PUSH PROGRAM

1. Files

Conceptually, the Push program has been developed to simulate JTLS pushing game data onto the internet as if the game were in progress. In practice, the program uses data files from a completed game, and transmits individual records from these files across the network to the Sun Workstation. The program sends these records in chronological order simulating sending these records as if they were being generated by the game. To accomplish this the time attribute from each record is determined and compared with the Push program generated clock. Additionally the order, domain and ranges of each record attribute must be determined and entered correctly on the Sun Workstation. This entailed a complete analysis of the data files as the first step in the development of the Push program. The Postprocessor User Guide [Ref. 4] contains an administrative listing of tables and attributes assembled in alphabetical order. Also the size of each numeric column is described in byte size, rather than the number of required characters as required by ORACLE. A comparison of the game files with the Postprocessor User Guide [Ref. 4] and the Data Requirements Manual [Ref. 13] furnished an assessment of attribute order and ranges.

Sixty of the sixty nine Postprocessor files are generated by the CEP and can be utilized by the Push program. The DATA_BASE, DICTIONARY, and DIRECTORY files contain administrative data and reside permanently on the Sun Workstation. The files COMBATSYS, REASONS, SUPPLIES, TARGETS, and UNITS are produced by the Scenario Preparation Program, and are used to match data produced by the CEP for report format purposes. The parameters of these data tuples can be manipulated by the game controller. These files should be transmitted to the Sun Workstation prior to the game. A subset of the CEP produced files associated with aircraft was selected for the demonstration of the Push program. The first step of the program was to create a corresponding time file for each data file to determine when a game record should be transmitted across the network. A time file was linked to the appropriate JTLS data file by a logical unit number. Each data file was assigned a unique logical unit. The numbers started at 10 to prevent utilizing a unit number reserved by the operating system. The corresponding time file was initially given the same logical unit assignment. However, in doing this the desired results were not obtained. For reasons unknown this method did not send data in the prescribed order. This was corrected by reassigning the time file the next sequential number of its linked relation data file. The maximum number that can be assigned to a logical device is ninety nine. As a result a maximum of forty five data files, and forty five time files can be used in this program. Each time file was created by using a definite iterative DO-loop. The loop control variable, UNT, was initiated to the value of the first logical unit number, increased by two, and terminated with the value of the last logical unit number. Each loop iteration invokes subroutine GETIME to create the time file. The data and associated time files are rewound for subsequent processing.

2. GETIME Routine

Subroutine GETIME reads the first thirteen characters of a data record into a character string to determine the elements of the time attribute. Data analysis revealed that the attribute TIME is the second attribute in all files, preceded by the attribute INTRVL. Although the range for INTRVL is 32767, interviews with JTLS contractors revealed the possibility that the length of INTRVL exceeding one character is remote. Further study of the data determined the minimum number of characters in attribute TIME is two, and the maximum number is ten. Therefore, only the first thirteen characters of each record were required to be read; one character for the interval, ten for the time, and two for delimiters "&".

The subroutine then examines elements of the array beginning with the third element (skips INTRVL and the first delimiter) until the second delimiter is located. The length of the attribute is then determined. Once the length is computed the required number of character concatenations is determined and executed.

The concatenation process appears to be rather awkward as written. It was originally implemented as a definite iterative DO-loop structure. For reasons unknown, the loop did not concatenate the array elements, and a series of IF-THEN decisions were implemented in lieu of the loop. Finally, the subroutine reclassifies TIME from a character variable to a real variable to allow comparison with the Push program generated timer.

Although each time and data file is in sequential order, records read from each file must be compared against each other to ensure data is sent across the network chronologically. An array type of real numbers, TMARY, is implemented to contain a time record from each of the time files. An array type of integers, UTARY, is implemented to contain the corresponding logical unit number of the time record.

The TMARY array is sorted in descending order, while the UTARY array mirrors the required sorting interchanges. This enables the program to order the logical unit of the time file which in turn points to the correct data record in the appropriate data file to be read and transmitted. The routine IARAYS places the first record of each time file and its corresponding logical unit number into TMARY and UTARY respectively. The variable INDEX is used to indicate array position. When the subroutine is finished, INDEX is assigned the value 1. Contriving this eliminates the need for two read routines. The RDTME routine is used both to initiate and update TMARY.

After the arrays are initialized, the records are sorted and transmitted until all of the files are empty. When a file is empty the subroutine COMPACT is invoked to decrement the counter of open files, increment the counter of closed files, and shift elements of the arrays to the left 1 unit. The counter for empty files is used to terminate the program when the counter is equal to the number of initially opened files. The counter for the remaining open files is used to provide the sort routine with the element comparison range and the elements in the left most position are no longer required because that file is empty. This increases the efficiency of the sort routine by eliminating unnecessary comparison.

3. <u>SORT Routine</u>

The sort routine selected for this program is a slight variation of the Bubblesort and uses the time records from the time files as objects of its sort. As previously stated all records within each individual file are already ordered, drastically reducing the number of required comparisons and interchanges. We found this sort routine to be the most efficient one to our knowledge for sorting an array that is predominantly arranged in order. This variation of Bubblesort was contrasted with the original Bubblesort, Shakesort, Quicksort, and Heapsort and found to have the least number of comparisons and interchanges for an ordered array.

The outer loop in the sort routine sets the terminal condition, MORE, to false and adjusts the comparison range if required. The inner loop is used to make the comparisons and interchanges. If an interchange is not required on any iteration of the inner loop, the array is assumed sorted and the routine terminates. Upon completion of the sort routine the data file containing the next record for transmission is determined.

4. CHCLK Routine

After the correct data file has been identified it is then necessary to resolve when the record is to be read and transmitted. This is accomplished in subroutine CHCLK. When the value of the program clock is equal to the record clock the appropriate data record is read and sent through the network. If the times are not equal, the program clock is incremented, by 0.00001 until the times are equivalent. The program clock is compared with the time record after each increment.

The purpose of the program clock is simply to cause a delay in transmission utilizing a counter to represent a break in the data stream. This delay is simply in terms of the relative amount of time the game player would normally wait until that data set would appropriately be available for query purposes.

Prior to sending the data across the network, the data record is tagged with an identifier and delimiter in subroutine SNDREC. The tag permits the Pack program, on the Sun, to determine for which table the record destined. The actual data transmission is performed by utilizing the remote file access routine available in the FORTRAN Library on the VAX-11. Following data transmission the RDTME reads the next time record from the corresponding time file of the record just transmitted into the TMARY. The process continues until all the records have been sent across the network.

C. PULL PROGRAM

1. <u>Incoming Game Files</u>

The Pull program runs in the background mode timesharing with the PACK program, also running in the background mode. The Pull program places data which has been sent across the network by the Push program into one of two data files located on the Sun Workstation. These two data files designated GAME1.DAT and GAME2.DAT are selected to be written or read depending on the status of the global read lock (RLCK) and write lock (WLCK) variables.

2. <u>Read Lock, Write Lock</u>

Prior to accepting data from the VAX-11, the Pull program checks the status of the global variables RLCK8 or RLCK9 to see if GAME1.DAT or GAME2.DAT respectively is being read by the Pack program. If the read lock status is equal to 0 the file is available to be written into by the Pull program. Access by the Pack program to read a file is disabled by setting the write lock variables, WLCK8 or WLCK9, equal to 1. The Pull program then opens the file, accepts data from the VAX-11, writes data to the file, closes the file, and resets the write lock variable back to 0. This sequence is executed for each data set written into the file.

The Pull program alternates writing between GAME1.DAT and GAME2.DAT. The actual file that is selected to be written into depends on the RLCK status of the chosen file. This alternation of the files continues until GAME1.DAT is closed by the Push program. This closure signals the Pull program that there is no more data to be received at its end. The Pull program sets NOMO to TRUE, informing the Pack program that no more data are to be read while in the REPEAT-UNTIL loop.

D. THE PACK PROGRAM

1. Implementation

The Pack program initially attempts to read file GAME1.DAT and busy wait until WLCK8 is set to 0. Because WLCK8 is initialized to 1 in the Pull program, the Pack program will not access GAME1.DAT until at least one set of data is available to be read. Once access to GAME1.DAT is obtained by the Pack program, the RLCK8 variable is set to 1. This action denies access to GAME1.DAT by the Pull program. When all records have been processed by the Pack program, RLCK8 is set to 0 and access by the Pull program is again permitted. The Pack program will then busy wait to read GAME2.DAT until access is granted.

While reading GAME2.DAT during the last iteration, the Pull program could be writing for the last time into GAME1.DAT and set the variable NOMO to true. To read and process this data the Pack program will exit the REPEAT-UNTIL loop, open GAME1.DAT, read the stored data, and place the data into the proper table. When the EOF is read, the Pack program terminates. It is not necessary to perform this read procedure on GAME2.DAT because of the program structure.

2. Database Tables

When a record has been read from either GAME1.DAT or GAME2.DAT, it must be parsed, reassembled into its correct form, and finally placed into the appropriate JTLS table. The first step in this process is to extract the table identifier and attributes from the data record. Except the first and last items, the individual data items are delineated by the "&" delimiter on either side of the component. A space denotes the end of the record. The search for attributes continues until a space has been located, which sets the loop terminal condition, MORE, false and ends the process.

Each element of the character array RCD is examined by the program. If the character read is not a space or delimiter the variable LEN, used to identify the field length, is incremented. The next element of the array is then examined. If a delimiter is read, the subroutine CONCAN is called to concatenate the appropriate elements into a single character string variable ATTR. The variable POS is the beginning element location, and LEN determines the number of required concatenations. Upon return from subroutine CONCAN, subroutine CONVRT is called to determine which attribute the string represents.

When the variable ANUM is set to 1, the variable ATTR represents the table identifier. The table identifier is subsequently used with specific values of ANUM to determine placement of ATTR into the database tables. The variable ATTR is then transformed to the appropriate data type for insertion into the database table. The programs returns to subroutine FNDATT which prepares the variables ANUM, LEN, INX and POS for the next string.

Subroutine ENTTUP writes data into the appropriate table. The corresponding read lock variable is tested to gain control of the desired database table. If RLCK is equal 0, WLCK is assigned 1 to prohibit the query program from gaining access. The data is then written to the table, and WLCK is reassigned 0 to allow queries. If RLCK is equal to 1 the program will busy wait attempting to gain access until the query program sets RLCK to 0.

VIII. RELATIONAL DATABASE DESIGN

A relational database design has been included to facilitate the implementation of the ORACLE database on the Sun Workstation. A logical database design specifies the logical format of the database, the records to be maintained, their contents, and the relationships among those records [Ref 14]. The logical design is then transformed into a relational database design, which is compatible with ORACLE and contains detailed specifications of the data base structure [Ref 15]. For an in depth explanation on relational data base design, please refer to [Ref. 14], [Ref. 16] and [Ref. 17]. Comments on the relational data base design as they pertain to the Postprocessor are provided below.

The JTLS Postprocessor User Guide [Ref. 4], Data Requirements Manual [Ref. 13], and several sets of actual game data were used to construct the design. Our initial inspection of the JTLS User Guide [Ref. 18] and the game data revealed a number of irregularities and discrepancies. We did not attempt to improve or add to the tables but only to translate the narrative data available in the tables into a relational design format.

Comments on Relational Design:

1) Attributes are provided in the order they are produced by the game. During the logical design process order is normally unimportant, however attributes are provided in the order they are produced by the game. Order is included in this design, because it would have been impossible to construct the Push and Pull programs without it. Only after thoroughly analyzing several sets of game data could this determination be made as no source documentation is available.

2) There is a considerable amount of redundancy in the Postprocessor tables. Data items are included in a table even though they could be determined through some other table. The redundancy in the Postprocessor database is intended to speed query processing [Ref. 4].

IX. RESULT OF THE STUDY

A. INTRODUCTION

This thesis supports the Naval Postgraduate School's (NPS) interest in the study of the feasibility of improving the performance of JTLS by developing a method eliminating the Postprocessor. We recommend the use of a dedicated database system running on a separate machine, physically and logically interfaced to the VAX-11. The intent of this project was to determine the feasibility of improving the performance of JTLS by eliminating the Postprocessor. To this end, several software products were produced.

The NPS made available the VAX-11 complete with JTLS, the Sun Workstation, and the Sun microsystems' version of UNIX. The Postprocessor was neither examined nor executed because it was never available for our use. The choice of these specific hardware and software components was discussed in detail in Chapter 1. However, the primary reason remains that these items continue to support a wide variety of possible modifications which are seriously being contemplated to further enhance the performance of JTLS, to include a distributed gaming system scenario. As such, it was the goal of this thesis to define the boundaries of one small aspect of such a problem, and in turn develop as many deliverable products as possible in fulfillment of this goal.

B. SYNOPSIS

These programs were developed to simulate the data transmission from the VAX and the assimilation of the transmitted data into a JTLS database located on the Sun. The first product produced was the Push program. This algorithm mimics what is anticipated to be the function of the modified CEP. This modification will involve the creation of an interface between the game and the Ethernet. Next, a receiving algorithm had to be constructed which interfaced the Sun with the Ethernet. Finally, a program was developed which places the data into their respective database table.

The tables, which are accessed by the Pack program, have been scrutinized for possible points of improvement within the constraints as outlined in Chapter 1. After closely reviewing the written goals detailed by the proponents of JTLS, we determined that the tables will be left in their current configuration. However, no supporting documents for the construction of the tables could be found anywhere. This required that we design and develop the Relational Design from the available tables.

These database tables will need to be generated by the ORACLE database system. Once installed on the Sun, ORACLE must be initialized and configured to communicate with the Pack program. In this way input data will be accepted by ORACLE via a logical pipeline versus input generated from a keyboard.

To transport the data from the game to the ORACLE database system, the Sun Workstation will need to be included in the local area network (LAN) as configured in the war laboratory. This will necessitate that both the software and hardware installation take place as prescribed in the ORACLE technical manual. Not only is the Ethernet required, but also TCP/IP to permit the VAX to conduct its unidirectional data traffic.

C. ACCOMPLISHMENTS AND ANALYSIS

1. Push Program

The Push program has been completed and partially tested. Because there is not an internet connection between the VAX-11 and the Sun Workstation at this time, we can not prove conclusively that the data sent by the Push program will be received on the Sun as we anticipate. By including several debug statements throughout the program, the program executes the algorithm as outlined in Chapter 6. In the testing phase, the program was allowed to read several of the game files. It then performed the necessary processing to the read data and wrote the results into a file called GAME.DAT for our review. Without error, the data records had been selected, sorted and entered into the file in the correct time sequence. When more than one record had the same time stamp, the program took the first available record of that sequence and placed it into GAME.DAT. After fourteen different runs, the file GAME.DAT contained the exact same results.

2. Pull Program

The network interface aspects of the Pull program could not be tested, again because of the lack of the network facilities for the Sun. Though the program interacted correctly with the Pack program, no determination was made concerning its ability to interface with either the data generated by the Push program or the Ethernet.

Inconclusive testing of the Pull program was conducted by having the program read a static file titled GAME.DAT which resided on the Sun Workstation. Running in the background mode, the Pull program simulated data acceptance from the Push program. No artificial restrictions were imposed on how the Pull program read this file. The algorithm specified that it could read only one data item at a time, and must write that item into either GAME1.DAT or GAME2.DAT depending on the status of the respective RLCK. By manually setting RLCK of the respective files to 1 or 0, we were able to test whether the Pull program would respond correctly. Without error, the Pull program placed each data item into the correct file after determining that the other file RLCK had been set to 1.

To test the programs as exhaustively as possible, several boundary cases were also examined. Because of the construction of the algorithm, when both files were set to 1, it had no impact on the program. Since the program alternates writing to each file, it simply entered a busy wait state until the file it is waiting to write into is available. The program will not check the status of the other file until it has completed its write requirement to the first file. When both files have RLCK equal to 0, it again has no impact on the program. The Pull program will continue to write in the same file until a query access changes that RLCK from 0 to 1.

3. Pack Program

The Pack program and its interaction with the Pull program was extensively tested utilizing the two files GAME1.DAT and GAME2.DAT. The Pull program was run in the background mode and the Pack program in the foreground mode. No testing of the interface between the Pack program and ORACLE was possible at this time.

Though we are unable to verify the correctness of the compatibility of the Pack program with ORACLE, the results of the test involving the Pull program lends substantial credibility to the performance of the Pack program as outlined in Chapter 6.

Once the Pull program deposits its data into either GAME1.DAT or GAME2.DAT, the Pack program will be able to retrieve the data in the order the Push program sent them across the network. By running the Pack program in the foreground mode, we were able, with the aid of debug write-lines, to observe the program read each data entry and place the processed data into its proper attribute location. Because there was no database system to generate the initially empty tables, we are unable to prove beyond doubt if the format of the files are compatible with ORACLE. However, the ATTR data string was successfully parsed and each attribute item was received at its designated destination during testing.

Several boundary cases were examined to include

1) both files having their WLCK set to 1,
2) both files having their RLCK set to 0,
3) both files starting empty and finally,
4) the Pack program reading the last data item supplied by the Pull program.

All of these cases were handled without difficulty by the Pack program. If a string did not conform to the format we anticipated in the algorithm, the program has been designed to discard that string.

Concurrency control closely resembles the mutual exclusion problems which were a major area of research a few years ago [Ref. 17]. However, these two methods of time control differ in very subtle ways. In mutual exclusion we approach the problem from the program side, establishing critical sections and making sure no two critical sections are active at the same time. In concurrency control we make use of the lock variables RLCK and WLCK, approaching the problem from the data side, directly protecting each file, without regard to which piece of the program text is currently executing. When there are potentially many programs, such as Pull and Pack, that might access some of the same files, it makes more sense to put the access controls on the files, rather than in the programs. The implementation of the locks is also different because keeping a semaphore for each file on the file system for the unlikely event that someone might want to lock it is too expensive.

Automatic locking is often combined with atomic update in a form known as a transaction. In our algorithm the form of the transaction has been slightly modified, but the principle is the same. By definition a completed transaction includes the property of either running successfully to completion, or failing and leaving the system state unchanged. To run a transaction, one of the program locks the file that it is to use and blocks the other process from access. It can then read and write to that file. Before it is finished and the process releases the lock on the file, the process ensures that all changes have been permanently made to the file in a single atomic update. Until the changes are completed, no other process is able to see or access the file.

4. Relational Design

The JTLS tables described in the Postprocessor contained very little information on attribute order. A lengthy analysis and comparison of every CEP produced file, the Postprocessor User Guide [Ref. 4] and the Data Requirements Manual [Ref. 13] were necessary for the development of the Relational Database Design located in Appendix J. This process was extremely tedious and time consuming, but vital for the creation of the JTLS tables in the ORACLE DBMS and for the processing of the data through the Pack program. The Relational Database Design will save a substantial amount of time and effort in the continuation of this project.

D. PROJECTED IMPACT ON JTLS

As mentioned earlier, JTLS is a valid simulation offering a host of advantages usually outlined in literature such are [Ref. 1,2,3] over training exercises and actual combat. As with most things, minor improvements in appropriate areas may enhance the benefits of playing JTLS.

One such improvement is the implementation of a dedicated database management system which will run concurrently with the game. Although actual real-time analysis of gaming data by a sequentially executed program is virtually impossible [Ref. 17], getting as close as possible is the aim of this project.

The completion of the Relational Model allows for a more thorough analysis of all the database table requirements pertinent to this game. The developers of the Postprocessor have claimed that extensive attribute redundancy was a necessity to accommodate rapid data retrieval. With the addition of the ORACLE DBMS, this statement may need to be re-examined. This later version of the database system contains many dramatic improvements over the 1970's era DBMS' in the realm of speed and efficiency [Ref. 19].

The ultimate impact of this project on the future of JTLS is almost unlimited. Currently only a single central computer system is able to execute the game with several dummy terminals available for the players to enter their input. With the success of the LAN and the incorporation of a concurrently running data analysis capability, this system is placed one step closer to a truly distributed war game with major force component players dispersed over many miles.

To to keep this project within reasonable bounds, the proposal suggested the possibility of internetting only one Sun Workstation and a demonstration was to take place implementing this arrangement. However, it is obvious that a minimum of two such stations would need to be in place for query activity to take place for both sides of a two-sided combat situation. The number of Sun Workstations need not be restricted to the number of opponents, but rather to the number of MIPs involved in any single game. Then each player would have a dedicated source of information to support his decisions.

E. CONCLUSION

If it can be assumed that the developed programs satisfy their performance requirements, then it is possible to make several conclusions about the outcome of this feasibility test:

1) It is possible to use a dedicated processor for off-line retrieval of data from an operational system.

2) The inclusion of a Sun Workstation into the Ethernet will allow very close to real-time retrieval of pertinent data, in response to user SQL queries.

F. RECOMMENDATION

After close review of the completed products of this thesis, in conjunction with several conversations with those parties interested in JTLS, we recommend that a final attempt be made to actually internet Sun Workstations to the Ethernet for the exchange of game data. This is one part of the proposal that was not completed in terms of the actual conveyance of data from one computer system to another. Once the data is made available on the Sun, the ORACLE Technical Manual discusses the procedure to assimilate data received from a communications channel into the respective tables.

G. AREAS FOR FURTHER STUDY

Essentially there remain only two major areas of additional research required to complete this project within the scope and constraints outlined in this proposal. These include:

1) JTLS CODE MODIFICATION - For us, the SIMSCRIPT code for JTLS was not made available for review. A thorough and indepth study should be made on the practicality and the cost-benefits of modifying the algorithm to make it network compatible. Particular attention needs to be placed on that part of the code that involves the interface of the CEP to the Ethernet.

2) MENU DRIVEN QUERY - It is recommended that the menu driven interface be translated directly from the INGRES-based language to SQL*C. Though not difficult to implement, it is anticipated to be considerably large in scope. The menu driven queries appears well suited for command and control queries. Finally this will preclude the user from unnecessarily learning another aspect of an environment he may already consider hostile and unfriendly.

In conclusion, as a result of the comprehensive analysis of the problem and the development of the software programs and database designs, we are confident that the goal of this project is attainable. Those areas recommended for further actions should be pursued. We believe that the continuation of this project will produce the desired result.

APPENDIX A

PROGRAM FILES

ACAVAIL.DAT: JTLS file containing records on the number of aircraft available for launch.

ACAVAIL.SQL: Actual ORACLE database containing the data from ACAVAIL.DAT. Data is written to this file in the PACK program.

ACAVTM.DAT: Created by the PUSH program. Contains the TIME attribute of the records from ACAVAIL.DAT.

ACKILLED.DAT: JTLS file containing records of the number of aircraft killed.

ACKILLED.SQL: Actual ORACLE database containing the data from ACKILLED.DAT. Data is written to this file in the PACK program.

ACKLLTM.DAT: Created by the PUSH program. Contains the TIME attribute of the records from ACKILLED.DAT

ACLAUNCH.DAT: JTLS file containing records of the number of aircraft launched on missions.

ACLAUNCH.SQL: Actual ORACLE database containing the data from ACLAUNCH.DAT. Data is written to this file in the PACK program.

ACLAUTM.DAT: Created by the PUSH program. Contains the TIME attribute of the records from ACLAUNCH.DAT.

ACREM.DAT: JTLS file containing records on the number of aircraft on hand.

ACREM.SQL: Actual ORACLE database containing the data from ACREM.DAT. Data is written to this file in the PACK program.

ACRETM.DAT: Created by the PUSH program. Contains the TIME attribute of the records from ACREM.DAT.

GAME.DAT: Created by the PUSH program. This file is used to push data across the network. Created and written to in the PUSH program and read from in the PULL program.

GAME1.DAT & GAME2.DAT: Created by the PULL program. Data read from the PULL program is written to these files to be read by the PACK program. Two files are required to enable the PACK program to access data without interfering with the transmission of data across the network.

APPENDIX B

PUSH PROGRAM VARIABLES

CLKINC (CLOCK INCREMENT)

description: Real constant containing the value (".000001") to increment the simulated game clock.

utilization: Increments simulated game clock in subroutine CHCLK.

CTME (CHARACTER TIME)

description: Character variable containing the characters in the time attribute of data records.

utilization: Assigned the characters value of the time attribute in subroutine GETIME.

CUNIT (CHARACTER UNIT)

description: Character variable containing the character representation of the logical unit assigned to a data file on the Sun Workstation.

utilization: Determined and attached to the data record subroutine SNDREC. The value of CUNIT is used by the PACK program to determine which table to write record.

DELIM (DELIMITER)

description: Character constant containing the value "&". Attribute in JTLS records are separated by the ampersand.

utilization:

1) Determines when the last character of the time attribute has been located in subroutine GETIME.

2) Attached to the data record and sent across the network in subroutine SNDREC.

EFILES (EMPTY FILES)

description: Integer variable that contains the number of closed files.

utilization: (COMMON /BLK1/)

1) Determines terminal condition in MAIN program. When EFILES equals FILES program execution terminates.

2) Incremented in subroutine COMPACT.

FILES

description: Integer constant containing the number of JTLS input files used in this program.

utilization: Used to initialize OFILES in main program.

GMCLK (GAME CLOCK)

description:Real variable simulating the JTLS game time.

utilization:(COMMON /BLK2/)Determines when data is sent in subroutine SND.

INDEX

description: Integer variable containing the value of the index/position of the array TMARY.

utilization: (COMMON /BLK3/)

1) Increments the position of the arrays TMARY and UTARY during initialization in subroutine IARAYS.

2) Indicates what position of the array TMARY to place data in subroutine RDTME.

INFO

description: Character variable containing the characters of the file identifier and the data record.

utilization: Created and sent across the network in subroutine SNDREC.

ITEMP (INTEGER TEMPORARY)

description: Integer variable containing the value of an UTARY array element.

utilization: Used as temporary storage for the elements of array UTARY in subroutine SORT.

LENGTH

description: Integer variable containing the number of characters in a time attribute.

utilizations: Determines the number of character concatenations required in subroutine GETIME.

MAXPOS (MAXIMUM POSITION)

description: Integer constant containing the maximum number of characters in a record ("13") required to extract the time attribute.

utilization: Test if time attribute has too many characters in subroutine GETIME.

MINLEN (MINIMUM LENGTH)

description: Integer constant containing the minimum number of characters required ("2") in a time attribute.

utilization: Test if time attribute has enough characters in subroutine GETIME.

MORE

description: Logical variable.

utilization:

1) Terminates string comparison in subroutine GETIME.

2) Terminates sort in subroutine SORT.

3) Terminates clock increment in subroutine CKCLK.

OFILES (OPEN FILES)

description: Integer variable containing the number of data/time files that are open.

utilization: (COMMON /BLK4/)

1) Determines if sorting is required in MAIN program (Sorting is unnecessary, if one file is open).

2) Provides subroutine SORT with the number of elements, of the array TMARY, to be compared.

3) Decremented in subroutine COMPACT.

POSIT (POSITION)

description: Integer variable containing the value of the index of the character array STRING.

utilization: Positions the index for character array STRING, to determine the length of the time attribute in subroutine GETIME.

RECORD

description: Character variable containing the characters of a data file record.

utilization: Assigned the characters of the data record and attached to a file identifier in subroutine SNDREC.

RTEMP (REAL TEMPORARY)

description: Real variable containing the value of a TMARY array element.

utilization: Used as temporary storage for the elements of array TMARY in subroutine SORT.

RTME (REAL TIME)

description: Real variable containing the time attribute of a data record in decimal days.

utilization: Decoded from the character variable CTME and written to a time file in subroutine GETIME.

STRING

description: Character array containing the first thirteen characters of a data record.

utilization: The first thirteen characters of a data record are read into STRING. This enables the extraction of the time attribute in subroutine GETIME.

SRTPOS (START POSITION)

description: Integer constant containing the starting position ("3") of the characters in the time attribute of JTLS data records.

utilization: Initializes POSIT in subroutine GETIME.

TMARY (TIME ARRAY)

description: Real array that contains a record from each of the open time files.

utilization: (COMMON /BLK7/)

1) TMARY (INDEX) is assigned the value of the appropriate time file in subroutine RDTME.

2) Elements are compared and interchanged if necessary in subroutine SORT.

3) The first element of array TMARY (the time to send the next record) is compared with GMCLK to determine when the record will be sent in subroutine CKCLK.

4) Adjusted by shifting all elements to theleft in subroutine COMPACT.

UNT (UNIT)

description: Integer variable containing the value of the logical units of the data or time files.

utilization: (COMMON /BLK6/)

1) Used as counter in subroutine IFILES for creating time files.

2) The value of the logical data file unit to be read in subroutine GETIME.

3) Determines the value ("UNT+1") of the logical time file unit in subroutine GETIME.

4) Used as counter in subroutine IARAYS for initiating the arrays TMARY and UTARY.

5) Contains the value of the logical time file unit read in subroutine RDTME.

6) Contains the value of the logical time file unit to be closed in subroutine COMPACT.

7) Assigned the value of the array UTARY element INDEX (first array element) in MAIN program.

8) Determines the value ("UNT-1") of the logical data file unit of the data record to be read and sent across the network in subroutine SNDREC.

UTARY (UNIT ARRAY)

description: Integer array containing the values of logical time files unit numbers.

utilization: (COMMON /BLK7/)

1) Initialized in subroutine IARAYS.

2) The elements are interchanged as TMARY elements are interchanged in subroutine SORT.

3) Adjusted by shifting all elements to the left in subroutine COMPACT. (The value in the first element is the unit being closed).

APPENDIX C

PROGRAM PUSH CODE

```
      PROGRAM PUSH
      INTEGER EFILES,FILES,INDEX,OFILES,UNT,UTARY(45)
      REAL GMCLK,TMARY(45)
      COMMON /BLK1/EFILES,/BLK2/GMCLK,/BLK3/INDEX,/BLK4/OFILES
      COMMON /BLK5/TMARY,/BLK6/UNT,/BLK7/UTARY
      DATA EFILES,FILES,GMCLK,INDEX,TMARY/0,4,0.0,1,45*0.0/
      OFILES=FILES
      CALL IFILES
      CALL IARAYS
      INDEX=1
      PRINT*, 'SORTING AND SENDING DATA'
      DO WHILE(EFILES.NE.FILES)
         IF(OFILES.GT.1)CALL SORT
         UNT=UTARY(INDEX)
         CALL CHCLK
         CALL RDTME
      END DO
      CLOSE UNIT=7
      CLOSE UNIT=10
      CLOSE UNIT=12
      CLOSE UNIT=14
      CLOSE UNIT=16
      PRINT*, 'PROGRAM COMPLETED'
      STOP
      END
************************************************************
      SUBROUTINE IFILES
      INTEGER UNT
      COMMON /BLK6/UNT
      PRINT*,'OPENING FILES'
      OPEN(UNIT=7,FILE='GAME.DAT',STATUS='NEW')
      OPEN(UNIT=10,FILE='ACAVAIL.DAT,STATUS=OLD')
      OPEN(UNIT=11,FILE='ACAVTM.DAT',STATUS='NEW')
      OPEN(UNIT=12,FILE='ACKILLED.DAT,STATUS=OLD')
      OPEN(UNIT=13,FILE='ACKLLTM.DAT',STATUS='NEW')
      OPEN(UNIT=14,FILE='ACLAUNCH.DAT,STATUS=OLD')
      OPEN(UNIT=15,FILE='ACLAUTM.DAT',STATUS='NEW')
      OPEN(UNIT=16,FILE='ACREM.DAT,STATUS=OLD')
      OPEN(UNIT=17,FILE='ACRETM.DAT',STATUS='NEW')
      PRINT*,'CREATING TIME FILES'
      DO 10 UNT=10,16,2
           CALL GETIME
           REWIND UNT
           REWIND (UNT+1)
 10   CONTINUE
      PRINT*,'FINISH CREATING TIME FILES'
      RETURN
      END
************************************************************
      SUBROUTINE GETIME
      INTEGER LENGTH,MAXPOS,MINLEN,POSIT,SRTPOS,UNT
      REAL RTME
      CHARACTER*10 CTME,DELIM*1,STRING(13)*1
      LOGICAL MORE
      COMMON /BLK6/UNT
      DATA DELIM,MAXPOS,MINLEN,SRTPOS/'&',13,2,3/
 10   READ(UNT,100,END=103,ERR=102)STRING(I),I=1,MAXPOS
100   FORMAT(13A1)
      MORE=.TRUE.
      POSIT=SRTPOS
      LENGTH=0
      DO WHILE(MORE)
           IF(STRING(POSIT).EQ.DELIM)THEN
                MORE=.FALSE.
           ELSE
                 IF(LENGTH.EQ.MAXPOS.AND.STRING(POSIT).NE.
     1           DELIM)THEN
                     PRINT*,"TIME IS GREATER THAN F10.6
     1               FORMAT"
                     GO TO 103
                 ENDIF
                 LENGTH=LENGTH+1
                 POSIT=POSIT+1
           ENDIF
      END DO
      IF (LENGTH.LT.MINLEN)THEN
           PRINT*,"TIME IS ONLY ONE CHARACTER"
           GO TO 103
      ENDIF
      IF(LENGTH.EQ.2)CTME=STRING(3)//STRING(4)
      IF(LENGTH.EQ.3)CTME=STRING(3)//STRING(4)//STRING(5)
      IF(LENGTH.EQ.4)CTME=STRING(3)//STRING(4)//STRING(5)
     1//STRING(6)
      IF(LENGTH.EQ.5)CTME=STRING(3)//STRING(4)//STRING(5)
     1//STRING(6)//STRING(7)
      IF(LENGTH.EQ.6)CTME=STRING(3)//STRING(4)//STRING(5)
     1//STRING(6)//STRING(7)//STRING(8)
      IF(LENGTH.EQ.7)CTME=STRING(3)//STRING(4)//STRING(5)
     1//STRING(6)//STRING(7)//STRING(8)//STRING(9)
      IF(LENGTH.EQ.8)CTME=STRING(3)//STRING(4)//STRING(5)
     1//STRING(6)//STRING(7)//STRING(8)//STRING(9)//STRING(10)
      IF(LENGTH.EQ.9)CTME=STRING(3)//STRING(4)//STRING(5)
     1//STRING(6)//STRING(7)//STRING(8)//STRING(9)
     1//STRING(10)//STRING(11)
     IF(LENGTH.EQ.10)CTME=STRING(3)//STRING(4)//STRING(5)
     1//STRING(6)//STRING(7)//STRING(8)//STRING(9)
     1//STRING(10)//STRING(11)//STRING(12)
      DECODE(LENGTH,101,CTME)RTME
      WRITE((UNT+1),101)RTME
101   FORMAT(F10.6)
      GO TO 10
102    PRINT*,"ERROR IN READING DATA FILES SUBROUTINE GETIME
LINE 7"
103   RETURN
      END
************************************************************
      SUBROUTINE IARAYS
      INTEGER INDEX,UNT,UTARY(45)
      COMMON /BLK3/INDEX,/BLK6/UNT,/BLK7/UTARY
      DO 10 UNT=11,17,2
           CALL RDTME
           UTARY(INDEX)=UNT
           INDEX=INDEX+1
 10   CONTINUE
      RETURN
      END
************************************************************
      SUBROUTINE RDTME
      INTEGER INDEX,UNT
      REAL TMARY(45)
      COMMON /BLK3/INDEX,/BLK5/TMARY,/BLK6/UNT
      READ(UNT,100,END=101,ERR=102)TMARY(INDEX)
100   FORMAT(F10.6)
      RETURN
101   CALL COMPACT
      RETURN
102   PRINT*,"ERROR READING TIME FILES IN SUBROUTINE RDTME
     1LINE 5"
      RETURN
      END
*************************************************************
```

```
      SUBROUTINE COMPACT
      INTEGER EFILES,OFILES,UNT,UTARY(45)
      REAL TMARY(45)
      COMMON BLK1/EFILES,/BLK4/OFILES,/BLK5/TMARY,/BLK6/UNT,
     1/BLK7/UTARY
      J=OFILES-1
      IF(OFILES.GT.1)THEN
           DO 10 I=1,J
                TMARY(I)=TMARY(I+1)
                UTARY(I)=UTARY(I+1)
 10        CONTINUE
      ENDIF
      EFILES=EFILES+1
      OFILES=OFILES-1
      CLOSE UNIT=UNT
      PRINT*,('CLOSED UNIT',UNT)
      RETURN
      END
*************************************************************
      SUBROUTINE SORT
      INTEGER OFILES,ITEMP,UTARY(45)
      REAL RTEMP,TMARY(45)
      LOGICAL MORE
      COMMON /BLK4/OFILES,/BLK5/TMARY,/BLK7/UTARY
      J=OFILES-1
      MORE=.TRUE.
      DO 10 I=1,J
           IF(.NOT.MORE)RETURN
           K=OFILES-I
           MORE=.FALSE.
           DO 20 L=1,K
                IF(TMARY(L).GT.TMARY(L+1))THEN
                     MORE=.TRUE.
                     RTEMP=TMARY(L)
                     TMARY(L)=TMARY(L+1)
                               TMARY(L+1)=RTEMP
                     ITEMP=UTARY(L)
                     UTARY(L)=UTARY(L+1)
                     UTARY(L+1)=ITEMP
                ENDIF
 20        CONTINUE
 10   CONTINUE
      RETURN
      END
**************************************************************
      SUBROUTINE CHCLK
      INTEGER INDEX
      REAL CLKINC,GMCLK,TMARY(45)
      LOGICAL MORE
      COMMON /BLK2/GMCLK,/BLK3/INDEX,/BLK5/TMARY
      DATA CLKINC /.000001/
      MORE=.FALSE.
      DO WHILE(.NOT.MORE)
           IF(TMARY(INDEX).EQ.GMCLK)THEN
                CALL SNDREC
                MORE=.TRUE.
           ELSE
                GMCLK=GMCLK+CLKINC
           ENDIF
      END DO
      RETURN
      END
**************************************************************
      SUBROUTINE SNDREC
      INTEGER UNT
      CHARACTER*1 DELIM
      CHARACTER CUNIT*2,INFO*80,RECORD*77
      COMMON /BLK6/UNT
      DATA DELIM /'&'/
      IF(UNT.EQ.11)CUNIT='10'
      IF(UNT.EQ.13)CUNIT='11'
      IF(UNT.EQ.15)CUNIT='12'
      IF(UNT.EQ.17)CUNIT='13'
      READ((UNT-1),101,END=103,ERR=104)RECORD
101   FORMAT(A77)
      INFO=CUNIT//DELIM//RECORD
      WRITE (7,102)INFO
102   FORMAT (X,A80)
103   RETURN
104   PRINT*,"ERROR READING DATA FILE IN SUBROUTINE SNDREC
     1LINE 9"
      RETURN
      END
```

APPENDIX D

PROGRAM PUSH CODE DESCRIPTION

```
      PROGRAM PUSH
      INTEGER EFILES,FILES,INDEX,OFILES,UNT,UTARY(45)
      REAL GMCLK,TMARY(45)
      COMMON /BLK1/EFILES,/BLK2/GMCLK,/BLK3/INDEX,/BLK4/OFILES
      COMMON /BLK5/TMARY,/BLK6/UNT,/BLK7/UTARY
c
c Initialization of variables.
c
      DATA EFILES,FILES,GMCLK,INDEX,UTARY /0,4,0.0,1,45*0.0/
      OFILES=FILES
c
c Open data files and create time files.
c
      CALL IFILES
c
c Initialize arrays TMARY and UTARY
c
      CALL IARAYS
c
c Assign INDEX the value of "1". All subsequent reads of time
c files will be placed into the first element of array
c TMARY. See subroutine RDTME.
c
       INDEX=1
c
c End Initialization.
c
       PRINT*, 'SORTING AND SENDING DATA'
c
c Send records while data files are open. Terminate process
c if all data files are closed (The number of empty files is
c equal to the number of data files used).
c
      DO WHILE(EFILES.NE.FILES)
c
c Call subroutine SORT if there is more than one file open.
c Sorting is not required if only one file is open.
c
         IF(OFILES.GT.1)CALL SORT
c
c Assign the value of the first element of array UTARY to
c UNT.
c The  logical unit of the next record to be transmitted is
c "UNT-1". The logical unit of the next time file to be reads
c is the value "UNT".
c
         UNT=UTARY(INDEX)
c
c Determine when data should be sent.
c
         CALL CHCLK
c
c Get the next time file record.
c
         CALL RDTME
c
       END DO
c Close files.
c
       CLOSE UNIT=7
       CLOSE UNIT=10
       CLOSE UNIT=12
       CLOSE UNIT=14
       CLOSE UNIT=16
      PRINT*, 'PROGRAM COMPLETED'
      STOP
      END
************************************************************
      SUBROUTINE IFILES
c
c This subroutine opens data and time files. Logical
c assignments begin with number 10. The data files are
c assigned even units, and the time files are assigned a
c unit 1 higher of it's respective data file (the odd
c numbered units). The routine then call subroutine GETIME
c to create time files on the appropriate units.
c
      INTEGER UNT
      COMMON /BLK6/UNT
      PRINT*,'OPENING FILES'
c
c Open files and assign logical units.
c
c "GAME.DAT" is used to transmit data across the network.
c
      OPEN(UNIT=7,FILE='GAME.DAT',STATUS='NEW')
      OPEN(UNIT=10,FILE='ACAVAIL.DAT,STATUS=OLD')
      OPEN(UNIT=11,FILE='ACAVTM.DAT',STATUS='NEW')
      OPEN(UNIT=12,FILE='ACKILLED.DAT,STATUS=OLD')
      OPEN(UNIT=13,FILE='ACKLLTM.DAT',STATUS='NEW')
      OPEN(UNIT=14,FILE='ACLAUNCH.DAT,STATUS=OLD')
      OPEN(UNIT=15,FILE='ACLAUTM.DAT',STATUS='NEW')
      OPEN(UNIT=16,FILE='ACREM.DAT,STATUS=OLD')
      OPEN(UNIT=17,FILE='ACRETM.DAT',STATUS='NEW')
      PRINT*,'CREATING TIME FILES'
c
c This loop is for creating time files. The loop control
c variable "UNT" determines the data and time files to be
c read and written. Files are rewound for subsequent
c processing.
c
c
      DO 10 UNT=10,16,2
           CALL GETIME
           REWIND UNT
           REWIND (UNT+1)
 10   CONTINUE
      PRINT*,'FINISH CREATING TIME FILES'
      RETURN
      END
*************************************************************
c
      SUBROUTINE GETIME
c
c This routine extracts, and converts the characters of the
c time attribute to real format and writes them to a time
c file.
c
c Data analysis showed:
c
c     1) the third character of JTLS records is the first
c     character of the time attribute (SRTPOS).
c
c     2) the minimum number of characters in the time
c     attribute is two (MINLEN).
c
c      3) The maximun number of characters in a time attribute
c     is 10.
c
c     4) Attributes in JTLS records are separated by the
c     delimiter "&"(DELIM).
c
      INTEGER LENGTH,MAXPOS,MINLEN,POSIT,SRTPOS,UNT
      REAL RTME
      CHARACTER*10 CTME,DELIM*1,STRING(13)*1
      LOGICAL MORE
      COMMON /BLK6/UNT
      DATA DELIM,MAXPOS,MINLEN,SRTPOS/'&',13,2,3/
c
c Read the first thirteen characters of a record into the
c character array STRING. The subroutine will return when an
c end of file is read.
c
 10   READ(UNT,100,END=103,ERR=102)STRING(I),I=1,MAXPOS
100   FORMAT(13A1)
      MORE=.TRUE.
      POSIT=SRTPOS
      LENGTH=0
c
c Determine the number of characters in the time attribute.
c When the delimiter "&" has been read the number of
c characters in the time attribute is determined.
c
      DO WHILE(MORE)
c
c Check string position for delimiter.
c
           IF(STRING(POSIT).EQ.DELIM)THEN
                MORE=.FALSE.
           ELSE
c
c Check for error. If the thirteenth position is checked and
c it is not the delimiter an error message will be provided.
c
                IF(LENGTH.EQ.MAXPOS.AND.STRING(POSIT).NE.
     1          DELIM)THEN
                     PRINT*,"TIME IS GREATER THAN F10.6
     1               FORMAT"
                     GO TO 103
                ENDIF
c
c Increment length and string position.
c
                LENGTH=LENGTH+1
                POSIT=POSIT+1
           ENDIF
      END DO
c
c If length is less than two characters an error message will
c result.
c
      IF (LENGTH.LT.MINLEN)THEN
           PRINT*,"TIME IS ONLY ONE CHARACTER"
           GO TO 103
      ENDIF
c
c After the number of characters in the time attribute have
c been determined, the following decision statements will
c determine
c the  appropriate number of concatenations to be made.
c
      IF(LENGTH.EQ.2)CTME=STRING(3)//STRING(4)
      IF(LENGTH.EQ.3)CTME=STRING(3)//STRING(4)//STRING(5)
      IF(LENGTH.EQ.4)CTME=STRING(3)//STRING(4)//STRING(5)
     1//STRING(6)
      IF(LENGTH.EQ.5)CTME=STRING(3)//STRING(4)//STRING(5)
     1//STRING(6)//STRING(7)
      IF(LENGTH.EQ.6)CTME=STRING(3)//STRING(4)//STRING(5)
     1//STRING(6)//STRING(7)//STRING(8)
      IF(LENGTH.EQ.7)CTME=STRING(3)//STRING(4)//STRING(5)
     1//STRING(6)//STRING(7)//STRING(8)//STRING(9)
      IF(LENGTH.EQ.8)CTME=STRING(3)//STRING(4)//STRING(5)
     1//STRING(6)//1STRING(7)//STRING(8)//STRING(9)
     1//STRING(10)
```

```
      IF(LENGTH.EQ.9)CTME=STRING(3)//STRING(4)//STRING(5)
     1//STRING(6)//STRING(7)//STRING(8)//STRING(9)
     1//STRING(10)//STRING(11)
      IF(LENGTH.EQ.10)CTME=STRING(3)//STRING(4)//STRING(5)
     1//STRING(6)//STRING(7)//STRING(8)//STRING(9)
     1//STRING(10)//STRING(11)//STRING(12)
c
c Convert the time attribute from character to real, and
c write it to the appropriate time file.
c
      DECODE(LENGTH,101,CTME)RTME
      WRITE((UNT+1),101)RTME
101   FORMAT(F10.6)
c
c Control is returned to statement 10 to read the next data
c file record.
c
      GO TO 10
102   PRINT*,"ERROR IN READING DATA FILES SUBROUTINE GETIME
  1LINE 7"
103   RETURN
      END
************************************************************
      SUBROUTINE IARAYS
c
c This subroutine is used to initialize arrays TMARY and
c UTARY. The first time record of each time file is read
c into successive elements of TMARY. The elements of array
c UTARY contain the logical unit value of the corresponding
c elements in array TMARY.
c
      INTEGER INDEX,UNT
      COMMON /BLK3/INDEX,/BLK6/UNT
c
c The time files are the odd logical units beginning with
c unit "11". The first record of each time file is read into
c successive elements of array TMARY.
c
      DO 10 UNT=11,17,2
           CALL RDTME
           INDEX=INDEX+1
           UTARY(INDEX)=UNT
10    CONTINUE
      RETURN
      END
************************************************************
      SUBROUTINE RDTME
c
c This routine is used to read a time record into the
c appropriate element of TMARY. If an end of file is read,
c subroutine COMPACT is called to adjust the arrays and
c close the empty file.
c
      INTEGER INDEX,UNT
      REAL TMARY(45)
      COMMON /BLK3/INDEX,/BLK5/TMARY,/BLK6/UNT
      READ(UNT,100,END=101,ERR=102)TMARY(INDEX)
100   FORMAT(F10.6)
      RETURN
101   CALL COMPACT
      RETURN
102   PRINT*,"ERROR READING TIME FILES IN SUBROUTINE RDTME
     1LINE 5"
      RETURN
      END
*************************************************************
      SUBROUTINE COMPACT
c
c This routine is called when a end of file indicator has
c been read.The routine will shift elements of the arrays
```

```
c TMARY, and UTARY to the left writing over the first
c element of each array. This is done to reduce the number
c of comparisons required in the sort routine.
c
c
      INTEGER EFILES,OFILES,UNT,UTARY(45)
      REAL TMARY(45)
      COMMON /BLK1/EFILES,/BLK4/OFILES,/BLK5/TMARY,
     1/BLK6/UNT,/BLK7/UTARY
c
      J=OFILES-1
c
c Adjustment is not required if one file is open.
c
     IF(OFILES.GT.1)THEN
           DO 10 I=1,J
                TMARY(I)=TMARY(I+1)
                UTARY(I)=UTARY(I+1)
 10        CONTINUE
      ENDIF
c
c Increment the number of empty files, decrement the number
c of open files, and close the unit.
c
      EFILES=EFILES+1
      OFILES=OFILES-1
      CLOSE UNIT=UNT
      PRINT*,('CLOSED UNIT',UNT)
      RETURN
      END
****************************************************************
      SUBROUTINE SORT
c
c This routine sorts the array TMARY in descending order.
c Interchanges of elements of TMARY necessitates an
c interchange of the corresponding elements in UTARY.
c
      INTEGER OFILES,ITEMP,UTARY(45)
      REAL RTEMP,TMARY(45)
      LOGICAL MORE
      COMMON /BLK4/OFILES,/BLK5/TMARY,/BLK7/UTARY
      J=OFILES-1
      MORE=.TRUE.
c
c The program will return if an interchange is not required
c during any iteration of the following loop.
c
      DO 10 I=1,J
           IF(.NOT.MORE)RETURN
c
c Adjust the number of required comparisons.
c
           K=OFILES-I
           MORE=.FALSE.
c
c This loop places the greatest value compared into array
c element K. Upon completion of this loop it is no longer
c necessary to compare position K.
c
           DO 20 L=1,K
                IF(TMARY(L).GT.TMARY(L+1))THEN
                     MORE=.TRUE.
                     RTEMP=TMARY(L)
                     TMARY(L)=TMARY(L+1)
                     TMARY(L+1)=RTEMP
                     ITEMP=UTARY(L)
                     UTARY(L)=UTARY(L+1)
                     UTARY(L+1)=ITEMP
                ENDIF
 20        CONTINUE
 10   CONTINUE
      RETURN
      END
************************************************************
      SUBROUTINE CHCLK
C
C This routine determines when to send data across the
C network. A record will be sent when the record's time
C (contained in array TMARY) is equal to the simulated game
C clock. If the record's time is not equal to the simulated
C clock, the clock is incremented until they are equal.
C
      INTEGER INDEX
      REAL CLKINC,GMCLK,TMARY(45)
      LOGICAL MORE
      COMMON /BLK2/GMCLK,/BLK3/INDEX,/BLK5/TMARY
      DATA CLKINC /.000001/
      MORE=.FALSE.
      DO WHILE(.NOT.MORE)
C
C Compare record time with game clock. Send record if times
C are equal. Increment game clock if times are not equal.
C
          IF(TMARY(INDEX).EQ.GMCLK)THEN
                CALL SNDREC
                MORE=.TRUE.
           ELSE
                GMCLK=GMCLK+CLKINC
           ENDIF
      END DO
      RETURN
      END
*************************************************************
      SUBROUTINE SNDREC
C
C This routine reads a data record, combines the record with
C an identifier and writes the data to the file GAME.DAT.
C GAME.DAT is read by the PULL program on the Sun
C Workstation.
C
      INTEGER UNT
      CHARACTER*1 DELIM
      CHARACTER CUNIT*2,INFO*80,RECORD*77
      COMMON /BLK6/UNT
C
C Determine the table identifier to be attached to the
C record.
C
      IF(UNT.EQ.11)CUNIT='10'
      IF(UNT.EQ.13)CUNIT='11'
      IF(UNT.EQ.15)CUNIT='12'
      IF(UNT.EQ.17)CUNIT='13'
C
C Read data record.
C
      READ((UNT-1),101,END=103,ERR=104)RECORD
101   FORMAT(A77)
C
C Attach identifier to data record. The delimiter is required
C to distinguish the identifier and the first attribute of
C the  record.
C
      INFO=CUNIT//DELIM//RECORD
      WRITE (7,102)INFO
102   FORMAT (X,A80)
103   RETURN
104    PRINT*,"ERROR READING DATA FILE IN SUBROUTINE SNDREC
LINE 9"
      RETURN
      END
```

APPENDIX E

PULL & PACK PROGRAM VARIABLES

ANUM (ATTRIBUTE NUMBER)

description: Integer variable containing the number of attributes that have been extracted from the record.

utilization:

1) Determines how attributes obtain a value in subroutine CONVRT.

2) Initialized and incremented in subroutine FNDATT.

ATTR (ATTRIBUTE)

description: Character variable containing the value of a record's attributes.

utilization:

1) Assigned value in subroutine CONCAN.

2) Provides value to the actual attribute in subroutine CONVRT.

CONFLT (CONFLICT)

description: Integer variable containing the value of an actual table attribute.

utilization: (COMMON /BLK4/)

1) Assigned value in subroutine CONVRT.

2) Written to table in subroutine ENTTUP.

DEL (DELIMITER)

description: Charactercontaining the attribute delimiter "&".

utilization: Determines when the last character of an attribute has been located in subroutine FNDATT.

INFO

description: Character variable containing the information sent from the VAX-11.

utilization: Read from VAX and written to a data file on the Sun Workstation.

INX (INDEX)

description: Integer variable containing the index/position of the character array RCD.

utilization: Positions the index of character array RCD for comparison in determining the length of an attribute in subroutine FNDATT.

INTRVL (INTERVAL)

description: Integer variable containing the value of an actual table attribute.

utilization: (COMMON /BLK2/)

1) Assigned value in subroutine CONVRT.

2) Written to table in subroutine ENTTUP.

LEN (LENGTH)

description: Integer variable containing the number of characters in an attribute.

utilization:

1) Initialize and incremented in subroutine FNDATT.

2) Determines the number of required concatenation in subroutine CONCAN.

LMSN (LOSER MISSION)

description: Integer variable containing the value of an actual table attribute.

utilization: (COMMON /BLK2/)

1) Assigned value in subroutine CONVRT.

2) Written to table in subroutine ENTTUP.

MSN (MISSION)

description: Integer variable containing the value of an actual table attribute.

utilization: (COMMON /BLK2/)

1) Assigned value in subroutine CONVRT.

2) Written to table in subroutine ENTTUP.

NOMO

description: Logical variable.

utilization: Global variable shared between the Pull and Pack programs.

1) Set to true in program Pull when an EOF has been read from GAME.DAT.

2) Used to determine last iteration of REPEAT UNTIL loop in PACK program.

NUMWPN (NUMBER OF WEAPONS)

description: Integer variable containing the value of an actual table attribute.

utilization: (COMMON /BLK2/)

1) Assigned value in subroutine CONVRT.

2) Written to table in subroutine ENTTUP.

POS (POSITION)

description: Integer variable containing the value of an index in character array RCD. POS is the first position of an attribute.

utilization:

1) Assigned value in subroutine FNDATT.

2) Determines the position to start concatenations in subroutine CONCAN.

QUANT (QUANTITY)

description: Integer variable containing the value of an actual table attribute.

utilization: (COMMON /BLK2/)

1) Assigned value in subroutine CONVRT.

2) Written to table in subroutine ENTTUP.

RCD (RECORD)

description: Character array containing the data that is transmitted from the PUSH program.

utilization: (COMMON /BLK1/)

1) Read in main program.

2) Examined to determine and extract attributes in subroutine FNDATT.

3) Elements are concatenated to form attribute in subroutine CONCAN.

REASON

description: Integer variable containing the value of an actual table attribute.

utilization: (COMMON /BLK2/)

1) Assigned value in subroutine CONVRT.

2) Written to table in subroutine ENTTUP.

RLCK8 & RLCK9 (READ LOCK)

description: Integer variables containing the value 0 or 1.

utilization: RLCK8 and RLCK9 are global variables shared between the PULL and PACK programs to coordinate reading and writing between the two processes.

RLCK10 thru RLCK13 (READ LOCK)

description: Integer variables containing the value 0 or 1.

utilization: RLCK10 thru RLCK13 are global variables shared between the PACK and QUERY programs to coordinate reading and writing between the two processes.

SHTER (SHOOTER)

description: Character variable containing the value of an actual table attribute.

utilization: (COMMON /BLK4/)

1) Assigned value in subroutine CONVRT.

2) Written to table in subroutine ENTTUP.

SIDE

description: Character variable containing the value of an actual table attribute.

utilization: (COMMON /BLK4/)

1) Assigned value in subroutine CONVRT.

2) Written to table in subroutine ENTTUP.

SMSN

description: Integer variable containing the value of an actual table attribute.

utilization: (COMMON /BLK2/)

1) Assigned value in subroutine CONVRT.

2) Written to table in subroutine ENTTUP.

SP (SPACE)

description: Character constant containing the value ' '.

utilization: Determines the end of a data record in subroutine FNDATT.

TAG

description: Integer variable containing the value of a logical I/O unit from which record was sent.

utilization:

1) Determines actual attribute assignments in subroutine CONVRT.

2) Determines the table to which a data record (tuple) is entered in subroutine ENTTUP.

TIME

description: Real variable containing the value of an actual table attribute.

utilization: (COMMON /BLK3/)

1) Assigned value in subroutine CONVRT.

2) Written to table in subroutine ENTTUP.

UNIT

description: Character variable containing the value of an actual table attribute.

utilization: (COMMON /BLK4/)

1) Assigned value in subroutine CONVRT.

2) Written to table in subroutine ENTTUP.

WLCK8 & WLCK9 (WRITE LOCK)

description: Integer variables containing the value 0 or 1.

utilization: RLCK8 and RLCK9 are global variables shared between the PULL and PACK programs to coordinate reading and writing between the two processes.

WLCK10 thru WLCK13 (WRITE LOCK)

description: Integer variables containing the value 0 or 1.

utilization: WLCK10 thru WLCK13 are global variables shared between the PACK and QUERY programs to coordinate reading and writing between the two processes.

WPNTYP (WEAPON TYPE)

description: Integer variable containing the value of an actual table attribute.

utilization: (COMMON /BLK2/)

1) Assigned value in subroutine CONVRT.

2) Written to table in subroutine ENTTUP.

APPENDIX F

PROGRAM PULL CODE

```
      PROGRAM PULL
      INTEGER RLCK8,RLCK9,WLCK8,WLCK9
      CHARACTER INFO*80
      LOGICAL NOMO
      DATA NOMO,WLCK8,WLCK9/.FALSE.1,0/
      OPEN (UNIT=7,FILE='GAME.DAT',STATUS='OLD')
10    IF(RLCK8.EQ.0)THEN
           WLCK8=1
           OPEN (UNIT=8,FILE='GAME1.DAT',STATUS='NEW')
           READ (7,100,END=20) INFO
           WRITE(8,100)INFO
           CLOSE UNIT=8
           WLCK8=O
      ELSE
           WLCK9=1
           OPEN (UNIT=9,FILE='GAME2.DAT',STATUS='NEW')
           READ (7,100,END=20)INFO
           WRITE(9,100)INFO
           CLOSE UNIT=9
           WLCK9=0
      ENDIF
      GOTO 10
20    NOMO=.TRUE.
      END
```

APPENDIX G

PROGRAM PULL CODE DESCRIPTION

```
      PROGRAM PULL
c
c This Program reads data from the PUSH program on the VAX
c and writes data to either GAME1or GAME2. When the PACK
c program is reading data from GAME1 data is written to GAME2
c and when the PACK program is reading from GAME2 data is
c written to GAME1. This allows data to continually be
c received from the war game while the Sun processes data,
c and players query the database. Variables RLCK8, RLCK9,
c WLCK9, and  WLCK9 are global variables that allow the two
c processes to coordinate reading and writing.
c
      INTEGER RLCK8,RLCK9,WLCK8,WLCK9
      CHARACTER INFO*80
      LOGICAL NOMO
      DATA NOMO,WLCK8,WLCK9/.TRUE.,1,0/
      OPEN (UNIT=7,FILE='GAME.DAT',STATUS='OLD')
c
c If the Pack program is not reading from GAME1.DAT read data
c from the PUSH program and write data on GAME1.DAT.
c
 10   IF(RLCK8.EQ.0)THEN
c
c Disable pack program from reading GAME1.DAT.
c
              WLCK8=1
              OPEN (UNIT=8,FILE='GAME1.DAT',STATUS='NEW')
              READ (7,100,END=20) INFO
              WRITE(8,100)INFO
              CLOSE UNIT=8
c
c Enable Pack program to access GAME1.DAT.
c
              WLCK8=O
      ELSE
c
c If the Pack program is reading GAME1.DAT disable read from
c GAME2.DAT, read record and write data on GAME2.DAT.
c
              WLCK9=1
              OPEN (UNIT=9,FILE='GAME2.DAT',STATUS='NEW')
              READ (7,100,END=20)INFO
              WRITE(9,100)INFO
              CLOSE UNIT=9
              WLCK9=0
      ENDIF
      GOTO 10
c
c When an end of file is read from the Push Program set
c global variable NOMO to false. This informs the Pack
c program the game has terminated.
c
20   NOMO=.TRUE.
      END
```

APPENDIX H

PROGRAM PACK CODE

```
      PROGRAM PACK
      INTEGER  INTRVL,LMSN,MSN,NUMWPN,QUANT,REASON,SMSN,TAG,
      INTEGER WPNTYP RCLK8,RLCK9,RLCK10,RLCK11,RCLCK12,
      INTEGER RLCK13, WLCK8,WLCK9,WLCK10,WLCK11,WLCK12,WLCK13
      REAL TIME
      CHARACTER*1 RCD(80)
      CHARACTER CONFLT*20,SHTER*10,SIDE*4,UNIT*10
      LOGICAL NOMO
      COMMON /BLK1/RCD
      COMMON  /BLK2/INTRVL,LMSN,MSN,NUMWPN,QUANT,REASON,SMSN,
     &TAG,WPNTYP
      COMMON /BLK3/TIME
      COMMON /BLK4/CONFLT,SHTER,SIDE,UNIT
      COMMON  /BLK5/RLCK10,RLCK11,RLCK12,RLCK13,WLCK10,WLCK11,
     &WLCK12,WLCK13
      DATA RLCK8,RLCK9,WLCK10,WLCK11,WLCK12,WLCK13
     &/0,0,0,0,0,0/
      REPEAT
 10          IF(WLCK8.EQ.0)THEN
                  RLCK8=1
                  OPEN(UNIT=8,FILE='GAME1.DAT',STATUS='OLD')
 20               READ (8,100,END=30)(RCD(I)I=1,80)
                  CALL FNDATT
                  CALL ENTTUP
                  GOTO 20
             ELSE
                  GOTO 10
             ENDIF
  30         CLOSE UNIT=8
             RLCK8=0
  40         IF(WLCK9.E0.0)
                 RLCK9=1
                 OPEN(UNIT=9,FILE='GAME2.DAT',STATUS='OLD)
  50             READ (9,100,END=60)(RCD(I)I=1,80)
                 CALL FNDATT
                 CALL ENTTUP
                 GOTO 50
             ELSE
                 GOTO 40
             ENDIF
  60         CLOSE UNIT=9
             RLCK9=0
      UNTIL NOMO
      OPEN(UNIT=8,FILE='GAME1.DAT',STATUS='OLD')
 70   READ (8,100,END=80)(RCD(I)I=1,80)
      CALL FNDATT
      CALL ENTTUP
      GOTO 70
 80   CLOSE UNIT=8
 100  FORMAT(80A1)
      STOP
      END
*************************************************************
 SUBROUTINE FNDATT
      INTEGER ANUM,LEN,POS,INX
      CHARACTER*1 DELIM,RCD(80),SP
      CHARACTER ATTR*20
      COMMON/BLK1/RCD
      DATA ANUM,DELIM,INX,LEN,POS,SP/1,'&',1,0,1,' '/
 10   IF(RCD(INX).EQ.DELIM.OR.RCD(INX).EQ.SP)THEN
            IF(RCD(INX).EQ.SP)RETURN
            CALL CONCAN (POS,ATTR,LEN)
            CALL CONVRT (ANUM,ATTR,LEN)
            ANUM=ANUM+1
```

```
          LEN=0
          INX=INX+1
          POS=INX
      ELSE
          LEN=LEN+1
          INX=INX+1
          IF(INX.GT.80)PRINT*,'RECORD EXCEEDS 80 CHARACTERS'
      ENDIF
      GOTO 10
      END
*************************************************************
      SUBROUTINE CONCAN (I,ATTR,LEN)
      INTEGER I,LEN
      CHARACTER*1 RCD(80)
      CHARACTER ATTR*20
      COMMON /BLK1/ RCD
      IF(LEN.EQ.0)ATTR='O'
      IF(LEN.EQ.1)ATTR=RCD(I)
      IF(LEN.EQ.2)ATTR=RCD(I)//RCD(I+1)
      IF(LEN.EQ.3)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)
      IF(LEN.EQ.4)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
      IF(LEN.EQ.5)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)
      IF(LEN.EQ.6)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)
      IF(LEN.EQ.7)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)
      IF(LEN.EQ.8)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//IF(LEN.EQ.9)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)
     &//RCD(I+3)//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)
     &//RCD(I+8)
      IF(LEN.EQ.10)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)
     &//RCD(I+8)//RCD(I+9)
      IF(LEN.EQ.11)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)
     &//RCD(I+8)//RCD(I+9)//RCD(I+10)
      IF(LEN.EQ.12)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)//RCD(I+11)
      IF(LEN.EQ.13)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)//RCD(I+11)//RCD(I+12)
      IF(LEN.EQ.14)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)//RCD(I+11)//RCD(I+12)//RCD(I+13)
      IF(LEN.EQ.15)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)//RCD(I+11)//RCD(I+12)//RCD(I+13)
     &//RCD(I+14)
      IF(LEN.EQ.16)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)//RCD(I+11)//RCD(I+12)//RCD(I+13)
     &//RCD(I+14)//RCD(I+15)
      IF(LEN.EQ.17)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)//RCD(I+11)//RCD(I+12)//RCD(I+13)
     &//RCD(I+14)//RCD(I+15)//RCD(I+16)
      IF(LEN.EQ.18)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)//RCD(I+11)//RCD(I+12)//RCD(I+13)
     &//RCD(I+14)//RCD(I+15)//RCD(I+16)//RCD(I+17)
      IF(LEN.EQ.19)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)//RCD(I+11)//RCD(I+12)//RCD(I+13)
     &//RCD(I+14)//RCD(I+15)//RCD(I+16)//RCD(I+17)//RCD(I+18)
      IF(LEN.EQ.20)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)//RCD(I+11)//RCD(I+12)//RCD(I+13)
     &//RCD(I+14)//RCD(I+15)//RCD(I+16RCD(I+17)
     &//RCD(I+18)//RCD(I+19)
      IF(LEN.GT.20)PRINT*,‘ATTRIBUTE LENGTH GREATER THAN 20’
      RETURN
      END
****************************************************************
      SUBROUTINE CONVRT (ANUM,ATTR,LEN)
      INTEGER  ANUM,INTRVL,LEN,LMSN,MSN,NUMWPN,QUANT,REASON,
     &SMSN,TAG,WPNTYP
      REAL TIME
      CHARACTER ATTR*20,CONFLT*20,SHTER*10,SIDE*4,UNIT*10
      COMMON  /BLK2/INTRVL,LMSN,MSN,NUMWPN,QUANT,REASON,SMSN,
     &TAG,WPNTYP
      COMMON /BLK3/TIME
      COMMON /BLK4/CONFLT,SHTER,SIDE,UNIT
      IF(ANUM.EQ.1)DECODE(LEN,100,ATTR)TAG
      IF(ANUM.EQ.2)DECODE(LEN,101,ATTR)INTRVL
      IF(ANUM.EQ.3)DECODE(LEN,102,ATTR)TIME
      IF(ANUM.EQ.4)SIDE=ATTR
      IF(ANUM.EQ.5.AND.(TAG.EQ.10.OR.TAG.EQ.16))
     &DECODE(LEN,101,ATTR)QUANT
      IF(ANUM.EQ.5.AND.TAG.EQ.12)CONFLT=ATTR
      IF(ANUM.EQ.5.AND.TAG.EQ.14)DECODE(LEN,100,ATTR)MSN
      IF(ANUM.EQ.6)DECODE(LEN,101,ATTR)QUANT
      IF(ANUM.EQ.7)DECODE(LEN,100,ATTR)LMSN
      IF(ANUM.EQ.8)SHTER=ATTR
      IF(ANUM.EQ.9)DECODE(LEN,100,ATTR)SMSN
      IF(ANUM.EQ.10)DECODE(LEN,103,ATTR)WPNTYP
      IF(ANUM.EQ.11)DECODE(LEN,101,ATTR)NUMWPN
      IF(ANUM.EQ.12)DECODE(LEN,104,ATTR)REASON
      IF(ANUM.GT.13)PRINT*, ’NUMBER OF ATTRIBUTES EXCEEDS 12’
 100  FORMAT(I2)
 101  FORMAT(I5)
 102  FORMAT(F10.6)
 103  FORMAT(I4)
 104  FORMAT(I3)
      RETURN
      END
*****************************************************************
      SUBROUTINE ENTTUP
      INTEGER  INTRVL,LEN,LMSN,MSN,NUMWPN,QUANT,REASON,SMSN,
     &TAG,WPNTYP RLCK10,RLCK11,RLCK12,RLCK13,WLCK10,WLCK11,
     &WLCK12,WLCK13
      REAL TIME
      CHARACTER CONFLT*20,SHTER*10,SIDE*4,UNIT*10
      COMMON /BLK2/INTRVL,LMSN,MSN,NUMWPN,QUANT,REASON,SMSN,
     &TAG,WPNTYP
      COMMON /BLK3/TIME
      COMMON /BLK4/CONFLT,SHTER,SIDE,UNIT
      COMMON  /BLK5/RLCK10,RLCK11,RLCK12,RLCK13,WLCK10,WLCK11,
     &WLCK12,WLCK13
      IF(TAG.EQ.10)THEN
 10        IF(RLCK10.EQ.0)THEN
                WLCK10=1
                OPEN(UNIT=10,FILE='ACAVAIL.SQL',STATUS='OLD')
                WRITE(10,100)INTRVL,TIME,SIDE,UNIT,QUANT
                CLOSE UNIT=10
                WLCK10=0
           ELSE
                GOTO 10
           ENDIF
      ENDIF
      IF(TAG.EQ.11)THEN
 20        IF(RLCK11.EQ.0)THEN
                WLCK11=1
                OPEN(UNIT=11,FILE='ACKILLED.SQL',STATUS='OLD')
                WRITE(11,101)INTRVL,TIME,SIDE,UNIT,CONFLT,
     &          QUANT,LMSN,SHTER,SMSN,WPNTYP,NUMWPN,REASON
                CLOSE UNIT=11
                WLCK11=0
           ELSE
```

```
                GOTO 20
            ENDIF
        ENDIF
        IF(TAG.EQ.12)THEN
30          IF(RLCK12.EQ.0)THEN
                WLCK12=1
                OPEN(UNIT=12,FILE='ACLAUNCH.SQL',STATUS='OLD')
                WRITE(12,102)INTRVL,TIME,SIDE,UNIT,MSN,QUANT
                CLOSE UNIT=12
                WLCK12=0
            ELSE
                GOTO 30
            ENDIF
       ENDIF
       IF(TAG.EQ.13)THEN
40          IF(RLCK13.EQ.0)THEN
                WLCK13=1
                OPEN(UNIT=13,FILE='ACREM.SQL',STATUS='OLD')
                WRITE(13,100)INTRVL,TIME,SIDE,UNIT,QUANT
                CLOSE UNIT=13
                WLCK13=0
            ELSE
                GOTO 40
            ENDIF
       ENDIF
100    FORMAT(I5,F10.6,A4,A10,I5)
101    FORMAT(I5,F10.6,A4,A10,A20,I5,I2,A10,I2,I5,I3)
102    FORMAT(I5,F10.6,A4,A10,I2,I5)
       RETURN
       END
```

APPENDIX I

PROGRAM PACK CODE DESCRIPTION

```
      PROGRAM PACK
c
c The PACK program will read data from whichever file,
c GAME1.DAT or GAME2.DAT, that is not being written to by
c the PULL program. Once a file has been opened the PULL
c program is prevented from writing to the open file until
c all records have been read, deciphered, and c placed into
c the proper JTLS table. When an end of file has been read
c the file will be closed and made accessible to the PULL
c program.  The process continues alternating between files
c until the global variable is set to true by the Pull
c program.
c
      INTEGER  INTRVL,LMSN,MSN,NUMWPN,QUANT,REASON,SMSN,TAG,
     &WPNTYP,RCLK8,RLCK9,RLCK10,RLCK11,RLCK12,RLCK13,WLCK8,
     &WLCK9,WLCK10,WLCK11,WLCK12,WLCK13
      REAL TIME
      CHARACTER*1 RCD(80)
      CHARACTER CONFLT*20,SHTER*10,SIDE*4,UNIT*10
      LOGICAL NOMO
      COMMON /BLK1/RCD
      COMMON /BLK2/INTRVL,LMSN,MSN,NUMWPN,QUANT,REASON,SMSN,
     &TAG,WPNTYP
      COMMON /BLK3/TIME
      COMMON /BLK4/CONFLT,SHTER,SIDE,UNIT
      COMMON  /BLK5/RCLK10,RCLK11,RCLK12,RLCK13,WLCK10,WLCK11,
     &WLCK12,WLCK13
      DATA RLCK8,RLCK9,WLCK10,WLCK11,WLCK12,WLCK13
     &/0,0,0,0,0,0/
c
c Alternate reading files until global variable NOMO is true.
c
      REPEAT
c
c If Pull program is not writting to GAME1.DAT disable
c access, read, and process data.
c
 10         IF(WLCK8.EQ.0)THEN
                  RLCK8=1
                  OPEN(UNIT=8,FILE='GAME1.DAT,'STATUS='OLD')
 20               READ(8,100,END=30)(RCD(I)I=1,80)
c
c Process record by finding attributes and placing into
c appropriate tables.
c
                  CALL FNDATT
                  CALL ENTTUP
c
c Read file until end.
c
                  GOTO 20
            ELSE
c
c Attempt access until permitted.
c
                  GOTO 10
            ENDIF
c
c Close file and enable access to Pull program, when an end
c of file is read from GAME1.DAT.
c
 30         CLOSE UNIT=8
            RLCK8=0
c
c If Pull program is not writting to GAME2.DAT disable
c access, read, and process data.
```

```
c
 40        IF(WLCK9.EQ.0)THEN
                RLCK9=1
                OPEN(UNIT=9,FILE='GAME2.DAT','STATUS='OLD')
 50             READ(9,100,END=30)(RCD(I)I=1,80)
c
c Process record by finding attributes and placing into
c appropriate tables.
c
                CALL FNDATT
                CALL ENTTUP
c
c Read file until end.
c
                GOTO 50
           ELSE
c
c Attempt access until permitted.
c
                GOTO 40
           ENDIF
60         CLOSE UNIT=9
           RLCK9=0
c
c Process is repeated until NOMO is set to true.
c
      UNTIL NOMO
c
c While reading GAME2.DAT during the last iteration, the Pull
c program could be placing the final set of data into
c GAME1.DAT and set NOMO to true. This block of instruction
c outside the REPEAT UNTIL loop will ensure that GAME1.DAT
c will be read.
c
      OPEN(UNIT=8,FILE='GAME1.DAT,'STATUS='OLD')
 70   READ(8,100,END=80)(RCD(I)I=1,80)
c
c Process record by finding attributes and placing into
c appropriate tables.
c
      CALL FNDATT
      CALL ENTTUP
c
c Read file until end.
c
      GOTO 70
80    CLOSE UNIT=8
      STOP
100   FORMAT (80A1)
      END
*************************************************************
    SUBROUTINE FNDATT
c
c This subroutine determines the length and starting
c positions of attributes. When the end of an attribute
c (DEL) has been located subroutine CONCAN is called to
c concatenate the characters contained in the attribute.
c Subroutine CONVRT is called to DECODE and assign value to
c the actual table attributes. When a space has been located
c the rotine will terminate.
c
      INTEGER ANUM,LEN,POS,INX
      CHARACTER*1 DELIM,RCD(80),SP
      CHARACTER ATTR*20
      COMMON/BLK1/RCD
c
c Initialize variables
c
      DATA ANUM,DELIM,INX,LEN,POS,SP/1,'&',1,0,1,' '/
c
```

```
c The search for attributes continues until the last
c character is located.
c
 10   IF(RCD(INX).EQ.DELIM.OR.RCD(INX).EQ.SP)THEN
           IF(RCD(INX).EQ.SP)RETURN
c
c Concatenate characters of attribute.
c
           CALL CONCAN (POS,ATTR,LEN)
c
c Convert and/or assign actual attribute.
c
           CALL CONVRT (ANUM,ATTR,LEN)
c
c Increment attribute number and array index, initialize
c attribute length and assign the next attribute's starting
c position.
c
           ANUM=ANUM+1
           LEN=0
           INX=INX+1
           POS=INX
       ELSE
c
c Increment attribute length and array index if the end of an
c attribute or a record is not found. A error message will
c be printed if the record exceeds 80 characters.
c
           LEN=LEN+1
           INX=INX+1
           IF(INX.GT.80)PRINT*,'RECORD EXCEEDS 80 CHARACTERS'
       ENDIF
       GOTO 10
       END
************************************************************
      SUBROUTINE CONCAN (I,ATTR,LEN)
c
c This subroutine concatenates the characters elements of
c the attribute into a single character variable. The number
c of required concatenations is determined by the length of
c the attribute (LEN).
c
      INTEGER I,LEN
      CHARACTER*1 RCD(80)
      CHARACTER ATTR*20
      COMMON /BLK1/ RCD
c We were unable to confirm if all attributes were mandatory.
c We included a case for length zero for this possiblity.
c
      IF(LEN.EQ.0)ATTR='O'
      IF(LEN.EQ.1)ATTR=RCD(I)
      IF(LEN.EQ.2)ATTR=RCD(I)//RCD(I+1)
      IF(LEN.EQ.3)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)
      IF(LEN.EQ.4)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
      IF(LEN.EQ.5)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)
      IF(LEN.EQ.6)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)
      IF(LEN.EQ.7)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)
      IF(LEN.EQ.8)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)
      IF(LEN.EQ.9)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
      IF(LEN.EQ.10)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)
      IF(LEN.EQ.11)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)
```

```
      IF(LEN.EQ.12)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)//RCD(I+11)
      IF(LEN.EQ.13)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)//RCD(I+11)//RCD(I+12)
      IF(LEN.EQ.14)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)//RCD(I+11)//RCD(I+12)//RCD(I+13)
      IF(LEN.EQ.15)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)//RCD(I+11)//RCD(I+12)//RCD(I+13)
     &//RCD(I+14)
      IF(LEN.EQ.16)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)//RCD(I+11)//RCD(I+12)//RCD(I+13)
     &//RCD(I+14)//RCD(I+15)
      IF(LEN.EQ.17)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)//RCD(I+11)//RCD(I+12)//RCD(I+13)
     &//RCD(I+14)//RCD(I+15)//RCD(I+16)
      IF(LEN.EQ.18)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)//RCD(I+11)//RCD(I+12)//RCD(I+13)
     &//RCD(I+14)//RCD(I+15)//RCD(I+16)//RCD(I+17)
      IF(LEN.EQ.19)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)
     &//RCD(I+10)//RCD(I+11)//RCD(I+12)//RCD(I+13)
     &//RCD(I+14)//RCD(I+15)//RCD(I+16)//RCD(I+17)//RCD(I+18)
      IF(LEN.EQ.20)ATTR=RCD(I)//RCD(I+1)//RCD(I+2)//RCD(I+3)
     &//RCD(I+4)//RCD(I+5)//RCD(I+6)//RCD(I+7)//RCD(I+8)
     &//RCD(I+9)//RCD(I+10)//RCD(I+11)//RCD(I+12)//RCD(I+13)
     &//RCD(I+14)//RCD(I+15)//RCD(I+16)//RCD(I+17)//RCD(I+18)
     &//RCD(I+19)
      IF(LEN.GT.20)PRINT*,'ATTRIBUTE LENGTH GREATER THAN 20'
      RETURN
      END
****************************************************************
    SUBROUTINE CONVRT (ANUM,ATTR,LEN)
c
c This subroutine determines the table and tables actual
c attributes. Once the determination has been made the
c character value is decoded if required, and the value of
c the actual attribute is assigned.
c
      INTEGER ANUM,INTRVL,LEN,LMSN,MSN,NUMWPN,QUANT,REASON,
     &SMSN,TAG
      REAL TIME
      CHARACTER ATTR*20,CONFLT*20,SHTER*10,SIDE*4,UNIT*10
      COMMON /BLK2/INTRVL,LMSN,MSN,NUMWPN,QUANT,REASON,SMSN,
      &TAG,WPNTYP
      COMMON /BLK3/TIME
      COMMON /BLK4/CONFLT,SHTER,SIDE,UNIT
c
c The table identifier is always the first attribute. It is
c also used to determine other attribute assignments.
c
      IF(ANUM.EQ.1)DECODE(LEN,100,ATTR)TAG
c
c Identify and assign values to other attributes.
c
      IF(ANUM.EQ.2)DECODE(LEN,101,ATTR)INTRVL
      IF(ANUM.EQ.3)DECODE(LEN,102,ATTR)TIME
      IF(ANUM.EQ.4)SIDE=ATTR
      IF(ANUM.EQ.5.AND.(TAG.EQ.10.OR.TAG.EQ.16))
     &DECODE(LEN,101,ATTR)QUANT
      IF(ANUM.EQ.5.AND.TAG.EQ.12)CONFLT=ATTR
      IF(ANUM.EQ.5.AND.TAG.EQ.14)DECODE(LEN,100,ATTR)MSN
      IF(ANUM.EQ.6)DECODE(LEN,101,ATTR)QUANT
      IF(ANUM.EQ.7)DECODE(LEN,100,ATRR)LMSN
      IF(ANUM.EQ.8)SHTER=ATTR
      IF(ANUM.EQ.9)DECODE(LEN,103,ATTR)SMSN
      IF(ANUM.EQ.10)DECODE(LEN,103,ATTR)WPNTYP
      IF(ANUM.EQ.11)DECODE(LEN,101,ATTR)NUMWPN
      IF(ANUM.EQ.12)DECODE(LEN,104,ATTR)REASON
c
c The maximun number of attributes in any table is 12.
c An error message is printed is more than 12 attributes
c have been located.
c
      IF(ANUM.GT.12)PRINT*, 'NUMBER OF ATTRIBUTES EXCEEDS 12'
 100  FORMAT(I2)
 101  FORMAT(I5)
 102  FORMAT(F10.6)
 103  FORMAT(I4)
 104  FORMAT(I3)
      RETURN
      END
*************************************************************
      SUBROUTINE ENTTUP
c
c  This  subroutine  determines  which  table  to  enter  the
attributes into and then makes the entry.
c
      INTEGER INTRVL,LEN,LMSN,MSN,NUMWPN,QUANT,REASON,SMSN,
     &TAG,WPNTYP,RLCK10,RLCK11,RLCK12,RLCK,13,WLCK10,WLCK11,
     &WLCK12,WLCK13
      REAL TIME
      CHARACTER CONFLT*20,SHTER*10,SIDE*4,UNIT*10
      COMMON /BLK2/INTRVL,LMSN,MSN,NUMWPN,QUANT,REASON,SMSN,
     &TAG,WPNTYP
      COMMON /BLK3/TIME
      COMMON /BLK4/CONFLT,SHTER,SIDE,UNIT
      COMMON  /BLK5/RCLK10,RCLK11,RCLK12,RLCK13,WLCK10,WLCK11,
     &WLCK12,WLCK13
c
c The appropriate table is determined by the variable TAG.
c If the corresponding ORACLE database is accessible (not
c being queried), query access is disabled and the record is
c entered into the table. The file is then closed to enable
c queries. If the table is not accessible the program will
c attempt access until obtained.
c
c
     IF(TAG.EQ.10)THEN
 10       IF(RLCK10.EQ.0)THEN
               WCLK10=1
               OPEN(UNIT=10,FILE='ACAVAIL.SQL',STATUS='OLD')
               WRITE(10,100)INTRVL,TIME,SIDE,UNIT,QUANT
c
c Allow access by query programs after writting into table.
c
               CLOSE UNIT=10
               WCLK10=0
          ELSE
               GOTO 10
          ENDIF
      ENDIF
      IF(TAG.EQ.11)THEN
 20       IF(RLCK11.EQ.0)THEN
               WLCK11=1
               OPEN(UNIT=11,FILE='ACKILLED.SQL',STATUS='OLD')
               WRITE(11,101)INTRVL,TIME,SIDE,UNIT,CONFLT,
   &           QUANT,LMSN,SHTER,SMSN,WPNTYP,NUMWPN,REASON
c
c Allow access by query programs after writting into table.
c
               CLOSE UNIT=11
               WLCK11=0
          ELSE
               GOTO 20
          ENDIF
     ENDIF
    IF(TAG.EQ.12)THEN
 30       IF(RLCK12.EQ.0)THEN
               WLCK12=1
               OPEN(UNIT=12,FILE='ACLAUNCH.SQL',STATUS='OLD')
               WRITE(12,102)INTRVL,TIME,SIDE,UNIT,MSN,QUANT
c
c Allow access by query programs after writting into table.
c
               CLOSE UNIT=12
               WLCK12=0
          ELSE
               GOTO 30
          ENDIF
     ENDIF
     IF(TAG.EQ.13)THEN
 40       IF(RLCK13.EQ.0)THEN
               WLCK13=1
               OPEN(UNIT=13,FILE='ACREM.SQL',STATUS='OLD')
               WRITE(13,100)INTRVL,TIME,SIDE,UNIT,QUANT
               CLOSE UNIT=13
               WLCK13=0
c
c Allow access by query programs after writting into table.
c
          ELSE
               GOTO 40
          ENDIF
     ENDIF
 100 FORMAT(I5,F10.6,A4,A10,I5)
 101 FORMAT(I5,F10.6,A4,A10,A20,I5,I2,A10,I2,I5,I3)
 102 FORMAT(I5,F10.6,A4,A10,I2,I5)
     RETURN
     END
```

APPENDIX J

RELATIONS

```
ACAVAIL (INTRVL, TIME, SIDE, UNIT, QUANTITY)
       Key:  (INTRVL, TIME, UNIT)

ACKILLED (INTRVL, TIME, SIDE, UNIT, CONFLICT, QUANTITY,
L_MISSION, SHOOTER, S_MISSION, WPN_TYPE, NO_WPNS, REASON)
   Key:  (INTRVL, TIME, UNIT)

ACLAUNCH (INTRVL, TIME, SIDE, UNIT, MISSION, QUANTITY)
   Key:  (INTRVL, TIME, UNIT)

ACREM (INTRVL, TIME, SIDE, UNIT, QUANTITY)
   Key:  (INTRVL, TIME, UNIT)

CALIVE (INTRVL, TIME, SIDE, UNIT, QUANTITY)
   Key:  (INTRVL, TIME, UNIT)

CAVAIL (INTRVL, TIME, SIDE, UNIT, QUANTITY)
   Key:  (INTRVL, TIME, UNIT)

COMBATSYS (CODE, SIDE, CS)
   Key: CODE

CSATT (INTRVL, TIME, SIDE, UNIT, CS, REASON, QUANTITY)
   Key:  (INTRVL, TIME, UNIT, CS)

CSKV (INTRVL, TIME, SIDE, UNIT, VICTIM, KILLER, QUANTITY)
   Key:  (INTRVL, TIME, UNIT, VICTIM)

CSLOST (INTRVL, TIME, SIDE, UNIT, CS, REASON, QUANTITY)
   Key:  (INTRVL, TIME, UNIT, CS)

CSONHAND (INTRVL, TIME, SIDE, UNIT, CS, QUANTIY)
   Key:  (INTRVL, TIME, UNIT, CS)

CSRECD (INTRVL, TIME, SIDE, UNIT, CS, REASON, QUANTITY)
   Key:  (INTRVL, TIME, UNIT, CS)

DATA_BASE (NAME, CLASS)
   Key: NAME

DAYNIGHT (INTRVL, TIME, SUNUP)
   Key: (INTRVL, TIME)

DICTIONARY (TERM, TABLE, MEANING)
   Key: (TERM, TABLE)

DIRECTORY (TABLE, CONTENTS, EVENTS)
   Key: TABLE

HUMINT (INTRVL, TIME, SIDE, UNIT, QUANTITY)
   Key:  (INTRVL, TIME, UNIT)

MISSIONS (CODE, MISSION)
   Key: CODE

MSLFIRED (INTRVL, TIME, SIDE, UNIT, QUANTITY)
   Key:  (INTRVL, TIME, UNIT)

NAVARRNG  (INTRVL, TIME, SIDE, UNIT, AR_RNG)
   Key:  (INTRVL, TIME, UNIT)

NAVMSRNG (INTRVL, TIME, SIDE, UNIT, MS_RNG)
          Key:  (INTRVL, TIME, UNIT)
```

RELATIONS

```
NAVSPEED (INTRVL, TIME, SIDE, UNIT, SPEED)
   Key:  (INTRVL, TIME, UNIT)

NAVSRRNG (INTRVL, TIME, SIDE, UNIT, SR_RNG)
   Key:  (INTRVL, TIME, UNIT)

ORDERS   (INTRVL,   TIME_SENT,   SIDE,   ORDER_TYPE,   UNIT,
TIME_SPEC)
   Key:  (INTRVL, TIME_SENT, UNIT)

PPSTATUS (INTRVL, TIME, STATUS)
   Key:  (INTRVL, TIME)

REASONS (CODE, REASON)
   Key:  CODE

SCDEC (INTRVL, TIME, TARGET, CATEGORY, REASON, QUANTITY)
   Key:  (INTRVL, TIME, TARGET, CATEGORY)

SCDUEIN (INTRVL, TIME, SIDE, UNIT, CATEGORY, QUANTITY)
   Key:  (INTRVL, TIME, UNIT, CATEGORY)

SCDUEOUT (INTRVL, TIME, SIDE, UNIT, CATEGORY, QUANTITY)
   Key:  (INTRVL, TIME, UNIT, CATEGORY)

SCINC (INTRVL, TIME, TARGET, CATEGORY, REASON, QUANTITY)
   Key:  (INTRVL, TIME, TARGET, CATEGORY)

SCINDUMP (INTRVL, TIME, TARGET, CATEGORY, QUANTITY)
   Key:  (INTRVL, TIME, TARGET, CATEGORY)

SCLOST   (INTRVL,   TIME,   SIDE,   UNIT,   CATEGORY,   REASON,
QUANTITY, ACTION)
   Key:  (INTRVL, TIME, UNIT, CATEGORY)

SCONHAND (INTRVL, TIME, SIDE, UNIT, CATEGORY, QUANTITY)
   Key:  (INTRVL, TIME, UNIT, CATEGORY)

SCRECD   (INTRVL,   TIME,   SIDE,   UNIT,   CATEGORY,   REASON,
QUANTITY)
       Key:  (INTRVL, TIME, UNIT, CATEGORY)

SCSENT   (INTRVL,   TIME,   SIDE,   UNIT,   CATEGORY,   REASON,
QUANTITY)
   Key:  (INTRVL, TIME, UNIT, CATEGORY)

SCSHORT   (INTRVL,   TIME,   SIDE,   UNIT,   CATEGORY,   REASON,
QUANTITY)
   Key:  (INTRVL, TIME, UNIT, CATEGORY)

SCTRANS (INTRVL, TIME, SIDE, UNIT, REASON, DRY_WT, WET_WT)
   Key:  (INTRVL, TIME, UNIT)

SUPPLIES (CODE, SIDE, CATEGORY)
   Key: CODE

TALIVE (INTRVL, TIME, SIDE, UNIT, QUANTITY)
   Key:  (INTRVL, TIME, UNIT)

TARGETS (INTRVL, TIME, ID, NAME, CATEGORY)
   Key:  (INTRVL, TIME, ID)

TAVAIL (INTRVL, TIME, SIDE, UNIT, QUANTITY)
   Key:  (INTRVL, TIME, UNIT)

TGADA (INTRVL, TIME, ID, STATUS)
```

```
   Key:  (INTRVL, TIME, ID)

TGCAPABLE (INTRVL, TIME, ID, ACTION, REASON, PCT_CAPABLE)
   Key:  (INTRVL, TIME, ID)

TGDETECT (INTRVL, TIME, ID, SIDE, REASON)
   Key:  (INTRVL, TIME, ID)

TGRANGE (INTRVL, TIME, ID, RNG)
   Key:  (INTRVL, TIME, ID)

TGSIDE (INTRVL, TIME, ID, SIDE)
   Key:  (INTRVL, TIME, ID)

TGSIZE (INTRVL, TIME, ID, SIZE)
   Key:  (INTRVL, TIME, ID)

TGUNIT(INTRVL, TIME, ID, UNIT, LAT, LON)
   Key:  (INTRVL, TIME, ID)

TRKILLED (INTRVL, TIME, SIDE, UNIT, CARGOS, TANKERS, REASON)
   Key:  (INTRVL, TIME, UNIT)

UNITS (SHORT_NAME, LONG_NAME, TYPE, SUBTYPE, SIDE, AIRCRAFT)
   Key:  SHORT_NAME

UTADA (INTRVL, TIME, SIDE, UNIT, STATUS)
   Key:  (INTRVL, TIME, UNIT)

UTAIRBASE (INTRVL, TIME, SIDE, UNIT, AIRBASE) REASONS
   Key:  (INTRVL, TIME, UNIT)

UTARRIVES (INTRVL, TIME, SIDE, UNIT, LAT, LON)
   Key:  (INTRVL, TIME, UNIT)

UTCAS (INTRVL, TIME, SIDE, UNIT, SQUADRON, NO_AIRCRAFT)
   Key:  (INTRVL, TIME, UNIT)

UTCONTACT (INTRVL, TIME, UNIT1, UNIT2, STATUS, POSTURE1,
POSTURE2)
   Key:  (INTRVL, TIME, UNIT1,UNIT2)

UTDELAYED (INTRVL, TIME, SIDE, UNIT, DELAYER_SIDE, LAT, LON,
DURATION)
   Key:  (INTRVL, TIME, UNIT)

UTHQ (INTRVL, TIME, SIDE, UNIT, HQ, REASON)
   Key:  (INTRVL, TIME, UNIT)

UTINCAR (INTRVL, TIME, SIDE, UNIT, INC)
   Key:  (INTRVL, TIME, UNIT)

UTLIFTED (INTRVL, TIME, SIDE, UNIT, LIFTED, STATUS, REASON)
   Key:  (INTRVL, TIME, UNIT)

UTPOSTURE   (INTRVL,   TIME,   SIDE,   UNIT,   NEW_POSTURE,
OLD_POSTURE, LAT, LON)
   Key:  (INTRVL, TIME, UNIT)

UTREINF (INTRVL, TIME, SIDE, UNIT, REINFORCER, STATUS)
   Key:  (INTRVL, TIME, UNIT)

UTSTART  (INTRVL,  TIME,  SIDE,  UNIT,  LAT,  LON,  DEST_LAT,
DEST_LON)
   Key:  (INTRVL, TIME, UNIT)
```

RELATIONS

UTSTOP (INTRVL, TIME, SIDE, UNIT, LAT, LON, REASON)
Key: (INTRVL, TIME, UNIT)

UTSTRNGTH (INTRVL, TIME, SIDE, UNIT, STRENGTH)

Key: (INTRVL, TIME, UNIT)

UTSUPPORT (INTRVL, TIME, SIDE, UNIT, SUPPORT_UNIT, REASON)
Key: (INTRVL, TIME, UNIT)

UTTRANS (INTRVL, TIME, SIDE, UNIT, DRY_WT, WET_WT, REASON)
Key: (INTRVL, TIME, UNIT)

WEAPONS (CODE, TYPE, SIDE)
Key: CODE

WPNEXPEND (INTRVL, TIME, SIDE, UNIT, MISSION, QUANTITY, TYPE, REASON)
Key: (INTRVL, TIME, UNIT)

INTERRELATION CONTRAINTS

ACAVAIL [UNIT]	SUBSET OF UNIT [SHORT_NAME]
ACKILLED [UNIT]	SUBSET OF UNITS [SHORT_NAME]
ACKILLED [L_MISSION]	SUBSET OF MISSION [CODE]
ACKILLED [SHOOTER]	SUBSET OF UNITS [SHORT_NAME]
ACKILLED [S_MISSION]	SUBSET OF MISSION [CODE]
ACKILLED [WPN_TYPE]	SUBSET OF WEAPONS [CODE]
ACKILLED [NO_WPNS]	SUBSET OF WPNEXPED [QUANTITY]
ACKILLED [REASON]	SUBSET OF REASONS [CODE]
ACLAUNCH [UNIT]	SUBSET OF UNITS [SHORT_NAME]
ACLAUNCH [MISSION]	SUBSET OF MISSION [CODE]
ACREM [UNIT]	SUBSET OF UNITS [SHORT_NAME]
CALIVE [UNIT]	SUBSET OF UNITS [SHORT_NAME]
CAVAIL [UNIT]	SUBSET OF UNITS [SHORT_NAME]
CSATT [UNIT]	SUBSET OF UNITS [SHORT_NAME]
CSATT [CS]	SUBSET OF COMBATSYS [CS]
CSATT [REASON]	SUBSET OF REASONS [CODE]
CSKV [UNIT]	SUBSET OF UNITS [SHORT_NAME]
CSKV [VICTIM]	SUBSET OF COMBATSYS [CS]
CSKV [KILLER]	SUBSET OF COMBATSYS [CS]
CSLOST [UNIT]	SUBSET OF UNITS [SHORT_NAME]
CSLOST [REASON]	SUBSET OF REASONS [CODE]
CSONHAND [UNIT]	SUBSET OF UNITS [SHORT_NAME]
CSONHAND [CS]	SUBSET OF COMBATSYS [CS]
CSRECD [UNIT]	SUBSET OF UNITS [SHORT_NAME]
CSRECD [CS]	SUBSET OF COMBATSYS [CS]
CSRECD [REASON]	SUBSET OF REASONS [CODE]
HUMINT [UNIT]	SUBSET OF UNITS [SHORT_NAME]
MSLFIRED [UNIT]	SUBSET OF UNITS [SHORT_NAME]
NAVARRNG [UNIT]	SUBSET OF UNITS [SHORT_NAME]
NAVSPEED [UNIT]	SUBSET OF UNITS [SHORT_NAME]
NAVSRRNG [UNIT]	SUBSET OF UNITS [SHORT_NAME]
ORDERS [UNIT]	SUBSET OF UNITS [SHORT_NAME]
SCDEC [TARGET]	SUBSET OF TARGET [ID]

INTERRELATION CONTRAINTS

SCDEC [CATEGORY]	SUBSET OF SUPPLIES [CATEGORY]
SCDEC [REASON]	SUBSET OF REASONS [CODE]
SCDUEIN [UNIT]	SUBSET OF UNITS [SHORT_NAME]
SCDUEIN [CATEGORY]	SUBSET OF SUPPLIES [CATEGORY]
SCDUEOUT [UNIT]	SUBSET OF UNITS [SHORT_NAME]
SCDUEOUT [CATEGORY]	SUBSET OF SUPPLIES [CATEGORY]
SCINC [TARGET]	SUBSET OF TARGETS [ID]
SCINC [CATEGORY]	SUBSET OF SUPPLIES [CATEGORY]
SCINC [REASON]	SUBSET OF REASONS [CODE]
SCINDUMP [TARGET]	SUBSET OF TARGETS [ID]
SCINDUMP [CATEGORY]	SUBSET OF SUPPLIES [CATEGORY]
SCLOST [UNIT]	SUBSET OF UNITS [SHORT_NAME]
SCLOST [CATEGORY]	SUBSET OF SUPPLIES [CATEGORY]
SCONHAND [UNIT]	SUBSET OF UNITS [SHORT_NAME]
SCONHAND [CATEGORY]	SUBSET OF SUPPLIES [CATEGORY]
SCRECD [UNIT]	SUBSET OF UNITS [SHORT_NAME]
SCRECD [CATEGORY]	SUBSET OF SUPPLIES [CATEGORY]
SCRECD [REASON]	SUBSET OF REASONS [CODE]
SCSENT [UNIT]	SUBSET OF UNITS [SHORT_NAME]
SCSENT [CATEGORY]	SUBSET OF SUPPLIES [CATEGORY]
SCSENT [REASON]	SUBSET OF REASONS [CODE]
SCSHORT [UNIT]	SUBSET OF UNITS [SHORT_NAME]
SCSHORT [CATEGORY]	SUBSET OF SUPPLIES [CATEGORY]
SCSHORT [REASON]	SUBSET OF REASONS [CODE]
SCTRANS [UNIT]	SUBSET OF UNITS [SHORT_NAME]
SCTRANS [REASON]	SUBSET OF REASONS [CODE]
TALIVE [UNIT]	SUBSET OF UNITS [SHORT_NAME]
TAVAIL [UNIT]	SUBSET OF UNITS [SHORT_NAME]
TGADA [ID]	SUBSET OF TARGETS [ID]
TGCAPABLE [ID]	SUBSET OF TARGETS [ID]
TGCAPABLE [REASON]	SUBSET OF REASONS [CODE]
TGDETECT [ID]	SUBSET OF TARGETS [ID]
TGDETECT [REASON]	SUBSET OF REASONS [CODE]

INTERRELATION CONTRAINTS

TGRANGE [ID]	SUBSET OF TARGETS [ID]
TGSIDE [ID]	SUBSET OF TARGETS [ID]
TGSIZE [ID]	SUBSET OF TARGETS [ID]
TGUNIT [ID]	SUBSET OF TARGETS [ID]
TGUNIT [UNIT]	SUBSET OF UNITS [SHORT_NAME]
TRKILLED [UNIT]	SUBSET OF UNITS [SHORT_NAME]
UTADA [UNIT]	SUBSET OF UNITS [SHORT_NAME]
UTAIRBASE [UNIT]	SUBSET OF UNITS [SHORT_NAME]
UTAIRBASE [AIRBASE]	SUBSET OF UNITS [SHORT_NAME]
UTAIRBASE [REASON]	SUBSET OF REASONS [CODE]
UTARRIVES [UNIT]	SUBSET OF UNITS [SHORT_NAME]
UTCAS [UNIT]	SUBSET OF UNITS [SHORT_NAME]
UTCAS [SQUADRON]	SUBSET OF UNITS [SHORT_NAME]
UTCONTACT [UNIT1]	SUBSET OF UNITS [SHORT_NAME]
UTCONTACT [UNIT2]	SUBSET OF UNITS [SHORT_NAME]
UTDELAYED [UNIT]	SUBSET OF UNITS [SHORT_NAME]
UTHQ [UNIT]	SUBSET OF UNITS [SHORT_NAME]
UTHQ [HQ]	SUBSET OF UNITS [SHORT_NAME]
UTHQ [REASON]	SUBSET OF REASONS [CODE]
UTINCAR [UNIT]	SUBSET OF UNITS [SHORT_NAME]
UTINCAR [INC]	SUBSET OF UNITS [SHORT_NAME]
UTLIFTED [UNIT]	SUBSET OF UNITS [SHORT_NAME]
UTLIFTED [LIFTER]	SUBSET OF UNITS [SHORT_NAME]
UTPOSTURE [UNIT]	SUBSET OF UNITS [SHORT_NAME]
UTPOSTURE [REASON]	SUBSET OF REASONS [CODE]
UTREINF [UNIT]	SUBSET OF UNITS [SHORT_NAME]
UTREINF [REINFORCER]	SUBSET OF UNITS [SHORT_NAME]
UTSTART [UNIT]	SUBSET OF UNITS [SHORT_NAME]
UTSTOP [UNIT]	SUBSET OF UNITS [SHORT_NAME]
UTSTOP [REASON]	SUBSET OF REASONS [CODE]
UTSTRENGTH [UNIT]	SUBSET OF UNITS [SHORT_NAME]
UTSUPPORT [UNIT]	SUBSET OF UNITS [SHORT_NAME]
UTSUPPORT [SUPPORT_UNIT]	SUBSET OF UNITS [SHORT_NAME]

INTERRELATION CONTRAINTS

UTSUPPORT [REASON]	SUBSET OF REASONS [CODE]
UTTRANS [UNIT]	SUBSET OF UNITS [SHORT_NAME]
WPNEXPEND [UNIT]	SUBSET OF UNITS [SHORT_NAME]
WPNEXPEND [MISSION]	SUBSET OF MISSIONS [CODE]
WPNEXPEND [TYPE]	SUBSET OF WEAPONS [TYPE]
WPNEXPED [REASON]	SUBSET OF REASONS [CODE]

DATA DOMAINS

DOMAIN NAME	FORMAT & MEANING
AIR_RADAR_RANGE	Real number 0.0 to 100000. See Data Requirements Manual pp. A-37.
CLASSIFICATION	CHAR(20) Characters "#", and "&" are not allowed. Database security classification.
CODE_AIR_MISSION	Positive integer less than 17. See Postprocessor User Guide table C-19 pp. C-13.
CODE_AIR_WEAPON	Positive integer 1 to 9999. See Data Requirements Manual pp. A-274.
CODE_COMBAT_SYS	Positive integer 1 to 32767. (physical limitation).
CODE_SUPPLY_CATEGORY	Positive integer 2 to 32767 (physical limitation). See Data Requirements Manual pp. A-289.
CONFLICT	Value is "AIR.TO.AIR", "GROUND.TO.AIR", or "WHILE.NOT.FLYING".Identifies type of conflict for aircraft casualty.
EVENT	CHAR(15) Characters "#" and "&" are not allowed. Identifies event numbers feeding a table.
INTERVAL	Positive integer 1 to 32767. (physical limitation). Identifies a continuous period of game play.
LATITUDE	Real -90.0 to 90.0.
LONGITUDE	REAL -180.0 to 180.0.
MEANING	CHAR(100) Characters "#", and "&" are not allowed. Meaning of attribute, or contents of table.
NAME_AIR_MISSION	Values contained in Postprocessor User Guide table C-19, pp. C-13.
NAME_AIR_WEAPON	CHAR(20) Characters "#", and "&" are not allowed.
NAME_COMBAT_SYSTEM	CHAR(20) Character "#", or "&" are not allowed.
NAME_DATA_BASE	CHAR(20) Characters "#", and "&" are not allowed.
NAME_SUPPLY_CATEGORY	CHAR(20) Characters "#", or "&", are not allowed.
NAME_TABLE	CHAR(20) Characters "#", and "&" are not allowed.

DATA DOMAINS

DOMAIN NAME	FORMAT & MEANING
NAME_TARGET_SHORT	CHAR (20). Characters "#" and "&" are not allowed.
NAME_TARGET_LONG	CHAR(40). Characters "#" and "&" are not allowed.
NAME_TERM	CHAR(20) Characters "#", and "&" are not allowed. Attribute or column title in a table.
NAME_UNIT_SHORT	CHAR (10). Characters "#" and "&" are not allowed. Names of airbases, ground units, naval units, and squadrons.
NAME_UNIT_LONG	CHAR(20). Characters "#" and "&" are not allowed. Names of airbases, ground units, naval units, and squadrons.
NAVAL_MISSILE_RANGE	Real 0.0 to 1.7E38 (physical limitation). See Data Requirements Manual pp. A-262.
NAVAL_UNIT_SPEED	Real 0.001 to 1.7E38 (physical limitation). See Data Requirements Manual pp. A-452.
ORDER_TYPE	CHAR(40) Character "#", or "&" are not allowed. Identifies type of order given by player.
PERCENTAGE	Real number 0.0 to 1.0. identifies percentage of combat system attrited, combat system killed,combat system lost, combat system on hand, combat system received, and target capability.
QUANTITY	Positive integer 0 to 32767 (physical limitation). Identifies the number of aircraft in unit, aircraft killed, aircraft available available for launch, aircraft aircraft providing close air aircraft remaining, air weapons fired, cargo trucks killed, cargo trucks available for convoy, cargo trucks in convoys, cargo trucks remaining, HUMINT teams available, naval missiles fired, tanker trucks killed, tanker trucks available for convoy, tanker trucks in convoys, and tanker trucks remaining.
REASON_AIRBASE	Positive integer. Identifies method of establishing airbase.

DOMAIN NAME	FORMAT & MEANING
REASON_AIRCRAFT_KILL	Positive integer of value "6","7","63","64","65","66", "67","68","69", or "105". Identifies cause of aircraft attrition.
REASON_CODE	Positive integer 0 to 107. See Postprocessor User Guide table C-27, pp. C-17.
REASON_CS_LOST_CBT	Positive integer of value "6","8","10","11","42", or "50". Identifies cause of a combat system's attrition (combat).
REASON_CS_LOST_NCBT	Positive integer of value "24","25","40","43", or "44". Identifies cause of a combat system's attrition (non combat).
REASON_CS_RECEIVED	Positive integer of value "36","71", or "85". Identifies reason for receipt of combat system.
REASON_EXPEND	Positive integer of value "11", "15", or "50". Identifies reason for air weapon expended.
REASON_HQ	Positive integer of value "0","4","5","51", or 79". Identifies method of establishing headquarters.
REASON_NAME	CHAR(40) See values Postprocessor User Guide table C-27 pp. C-17.
REASON_SUPPLY_DECR	Positive integer of value "12", "13" or "14". Identifies reason for supply category decrease.
REASON_SUPPLY_INCR	Positive integer of value "26","27","28" or "29". Identifies reason for supply category increase.
REASON_SUPPLY_LOSS	Positive integer of value "4","5","16","19","26","27", "28","29","34","37","53","59", "60", or "61". Identifies reason for supply category lost.
REASON_SUPPLY_RECD	Positive integer of value "1","4","5","16","17","18", "19","34","37","41","53" "55","59","70",or "87". Identifies reason supply category received.
REASON_SUPPLY_SENT	Positive integer of value "4","5","16","19","26","27", "28","29","34","37","53","59",

DATA DOMAINS

DOMAIN NAME	FORMAT & MEANING
	"50","79",or "92". Identifies reason supply category sent.
REASON_SUPPLY_SHORT	Positive integer of value "12","13", or "14". Identifies reason for supply category shortage.
REASON_SUPPLY_TRANS	Positive integer of value "4","5", or "62". Identifies supply category means of transportation.
REASON_TARGET_CAP	Positive integer of value "0","6","7","9","26","27", "28","29","36","46","51" "78","84","93", or "107". Identifies reason for change in target status.
REASON_TARGET_DET	Positive integer of value "6","33","48","70","74", "78", or "93". Identifies source of target's detection.
REASON_TRUCK_KILL	Positive integer of value "6","7", or "9". Identifies cause of truck attrition.
REASON_UNIT_LIFT	Positive integer of value "4","5" or "62". Identifies method of unit lift.
REASON_UNIT_POSTURE	Positive integer of value "3","4","5","11","30","31", "32","40","50","51","52","62", "79", "80","82", "84", "86", "87","88" or "91". Identifies reason for change in unit posture.
REASON_UNIT_STOP	Positive integer of value "4","5","30","31","32","51", "82","86","87", or "95". Identifies reason for unit stop.
REASON_UNIT_SUPPORT	Positive integer of value "0","4","5","47","51" or "79". Identifies reason for unit support.
REASON_UNIT_TRANS	Positive integer of value "4","5", or "62". Identifies unit's means of transportation.
SIDE	CHAR(4) Value is "BLUE" or "RED". Identifies unit as friendly or hostile.
SIDES	CHAR(10) Value is "BLUE", "NEUTRAL", or "RED." Identifies targets and unit delays as friendly, hostile, or neutral.

DATA DOMAINS

DOMAIN NAME	FORMAT & MEANING
STATUS	Integer value of "0", or "1". Identifies status of day/night, Postprocessor, ADA of a target, ADA of an unit, unit lift, and unit reinforcement.
SUPPLY_CATEGORY_LOST	CHAR(20) Value is "CONSUMED", "ATTRITED", or "OTHER".
SURFACE_RADAR_RANGE	Real 0.0 to 1.7E38 (physical limitation). See Data Requirements Manual pp. A-264.
TARGET_CAPABILITY	CHAR(20) Value is "CREATED", "HIT", "KILLED", or "REPAIRED". Identifies the reason for a change in target capability.
TARGET_CATEGORY	CHAR(30) See values Postprocessor User Guide table C-41, pp. C-27.
TARGET_RANGE	Real 0.0 to 1.7E38 (physical limitation). See Data Requirements Manual pp. A-412.
TARGET_SIZE	Positive integer of value "1", "2", or "3". Identifies size of target. See Postprocessor User Guide table C-48, p. C-31.
TIME	Real number 0.0 to 365. Identifies game time (in decimal days).
UNIT_POSTURE	CHAR(20) See values Postprocessor Users Guide table C-56, pp. C-36.
UNIT_SUBTYPE	CHAR (20) See values Postprocessor Users Guide table C-51 pp. C-33.
UNIT_TYPE	CHAR (20) See values Postprocessor Users Guide table C-51 pp. C-33.
WEIGHT	Real number 0.0 to 1.7E38 (physical limitation). See Data Requirements Manual pp. A-373. Identifies weight for supply category or transported unit.

ATTRIBUTE DOMAIN CORRESPONDENCE

ATTRIBUTE	DOMAIN
ACAVAIL.INTRVL	INTERVAL
ACAVAIL.TIME	TIME
ACAVAIL.SIDE	SIDE
ACAVAIL.UNIT	NAME_UNIT_SHORT
ACAVAIL.QUANTITY	QUANTITY
ACKILLED.INTRVL	INTERVAL
ACKILLED.TIME	TIME
ACKILLED.SIDE	SIDE
ACKILLED.UNIT	NAME_UNIT_SHORT
ACKILLED.CONFLICT	CONFLICT
ACKILLED.QUANTITY	QUANTITY
ACKILLED.L_MISSION	CODE_AIR_MISSION
ACKILLED.SHOOTER	NAME_UNIT_SHORT
ACKILLED.S_MISSION	CODE_AIR_MISSION
ACKILLED.WPN_TYPE	CODE_AIR_WEAPON
ACKILLED.NO_WPNS	QUANTITY
ACKILLED.REASON	REASON_AIRCRAFT_KILL
ACLAUNCH.INTRVL	INTERVAL
ACLAUNCH.TIME	TIME
ACLAUNCH.SIDE	SIDE
ACLAUNCH.UNIT	NAME_UNIT_SHORT
ACLAUNCH.MISSION	CODE_AIR_MISSION
ACLAUNCH.QUANTITY	QUANTITY
ACREM.INTRVL	INTERVAL
ACREM.TIME	TIME
ACREM.SIDE	SIDE
ACREM.UNIT	NAME_UNIT_SHORT
ACREM.QUANTITY	QUANTITY
CALIVE.INTRVL	INTERVAL
CALIVE.TIME	TIME
CALIVE.SIDE	SIDE
CALIVE.UNIT	NAME_UNIT_SHORT

ATTRIBUTE DOMAIN CORRESPONDENCE

ATTRIBUTE	DOMAIN
CALIVE.QUANTITY	QUANTITY
CAVAIL.INTRVL	INTERVAL
CAVAIL.TIME	TIME
CAVAIL.SIDE	SIDE
CAVAIL.UNIT	NAME_UNIT_SHORT
CAVAIL.QUANTITY	QUANTITY
COMBATSYS.CODE	CODE_COMBAT_SYS
COMBATSYS.SIDE	SIDE
COMBATSYS.CS	NAME_COMBAT_SYSTEM
CSATT.INTRVL	INTERVAL
CSATT.TIME	TIME
CSATT.SIDE	SIDE
CSATT.UNIT	NAME_UNIT_SHORT
CSATT.CS	NAME_COMBAT_SYSTEM
CSATT.REASON	REASON_CS_LOST_CBT
CSATT.QUANTITY	PERCENTAGE
CSKV.INTRVL	INTERVAL
CSKV.TIME	TIME
CSKV.SIDE	SIDE
CSKV.UNIT	NAME_UNIT_SHORT
CSKV.VICTIM	NAME_COMBAT_SYSTEM
CSKV.KILLER	NAME_COMBAT_SYSTEM
CSKV.QUANTITY	PERCENTAGE
CSLOST.INTRVL	INTERVAL
CSLOST.TIME	TIME
CSLOST.SIDE	SIDE
CSLOST.UNIT	NAME_UNIT_SHORT
CSLOST.CS	NAME_COMBAT_SYSTEM
CSLOST.REASON	REASON_CS_LOST_NCBT
CSLOST.QUANTITY	PERCENTAGE
CSONHAND.INTRVL	INTERVAL
CSONHAND.TIME	TIME

ATTRIBUTE DOMAIN CORRESPONDENCE

ATTRIBUTE	DOMAIN
CSONHAND.SIDE	SIDE
CSONHAND.UNIT	NAME_UNIT_SHORT
CSONHAND.CS	NAME_COMBAT_SYSTEM
CSONHAND.QUANTITY	PERCENTAGE
CSRECD.INTRVL	INTERVAL
CSRECD.TIME	TIME
CSRECD.SIDE	SIDE
CSRECD.UNIT	NAME_UNIT_SHORT
CSRECD.CS	NAME_COMBAT_SYSTEM
CSRECD.REASON	REASON_CS_RECIEVED
CSRECD.QUANTITY	PERCENTAGE
DATA_BASE.NAME	NAME_DATA_BASE
DATA_BASE.CLASS	CLASSIFICATION
DAYNIGHT.INTRVL	INTERVAL
DAYNIGHT.TIME	TIME
DAYNIGHT.SUNUP	STATUS
DICTIONARY.TERM	NAME_TERM
DICTIONARY.TABLE	NAME_TABLE
DICTIONARY.MEANING	MEANING
DIRECTORY.TABLE	NAME_TABLE
DIRECTORY.CONTENTS	MEANING
DIRECTORY.EVENTS	EVENT
HUMINT.INTRVL	INTERVAL
HUMINT.TIME	TIME
HUMINT.SIDE	SIDE
HUMINT.UNIT	NAME_UNIT_SHORT
HUMINT.QUANTITY	QUANTITY
MISSIONS.CODE	CODE_AIR_MISSION
MISSIONS.MISSION	NAME_AIR_MISSION
MSLFIRED.INTRVL	INTERVAL
MSLFIRED.TIME	TIME
MSLFIRED.SIDE	SIDE

ATTRIBUTE DOMAIN CORRESPONDENCE

ATTRIBUTE	DOMAIN
MSLFIRED.UNIT	NAME_UNIT_SHORT
MSLFIRED.QUANTITY	QUANTITY
NAVARRNG.INTRVL	INTERVAL
NAVARRNG.TIME	TIME
NAVARRNG.SIDE	SIDE
NAVARRNG.UNIT	NAME_UNIT_SHORT
NAVARRNG.AR_RNG	AIR_RADAR_RANGE
NAVMSRNG.INTRL	INTERVAL
NAVMSRNG.TIME	TIME
NAVMSRNG.SIDE	SIDE
NAVMSRNG.UNIT	NAME_UNIT_SHORT
NAVMSRNG.MS_RNG	NAVAL_MS_RANGE
NAVSPEED.INTRVL	INTERVAL
NAVSPEED.TIME	TIME
NAVSPEED.SIDE	SIDE
NAVSPEED.UNIT	NAME_UNIT_SHORT
NAVSPEED.SPEED	NAVAL_UNIT_SPEED
NAVSRRNG.INTRVL	INTERVAL
NAVSRRNG.TIME	TIME
NAVSRRNG.SIDE	SIDE
NAVSRRNG.UNIT	NAME_UNIT_SHORT
NAVSRRNG.SR_RNG	SURFACE_RADAR_RANGE
ORDERS.INTRVL	INTERVAL
ORDERS.TIME_SENT	TIME
ORDERS.SIDE	SIDE
ORDERS.ORDER_TYPE	ORDER_TYPE
ORDERS.UNIT	NAME_UNIT_SHORT
ORDERS.TIME_SPEC	TIME
PPSTATUS.INTRVL	INTERVAL
PPSTATUS.TIME	TIME
PPSTATUS.STATUS	STATUS
REASONS.CODE	REASON_CODE

ATTRIBUTE DOMAIN CORRESPONDENCE

ATTRIBUTE	DOMAIN
REASONS.NAME	REASON_NAME
SCDEC.INTRVL	INTERVAL
SCDEC.TIME	TIME
SCDEC.TARGET	NAME_TARGET_SHORT
SCDEC.CATEGORY	NAME_SUPPLY_CATEGORY
SCDEC.REASON	REASON_SUPPLY_DECR
SCDEC.QUANTITY	WEIGHT
SCDUEIN.INTRVL	INTERVAL
SCDUEIN.TIME	TIME
SCDUEIN.SIDE	SIDE
SCDUEIN.UNIT	NAME_UNIT_SHORT
SCDUEIN.CATEGORY	NAME_SUPPLY_CATEGORY
SCDUEIN.QUANTITY	WEIGHT
SCDUEOUT.INTRVL	INTERVAL
SCDUEOUT.TIME	TIME
SCDUEOUT.SIDE	SIDE
SCDUEOUT.UNIT	NAME_UNIT_SHORT
SCDUEOUT.CATEGORY	NAME_SUPPLY_CATEGORY
SCDUEOUT.QUANTITY	WEIGHT
SCINC.INTRVL	INTERVAL
SCINC.TIME	TIME
SCINC.SIDE	SIDE
SCINC.TARGET	NAME_TARGET_SHORT
SCINC.CATEGORY	NAME_SUPPLY_CATEGORY
SCINC.REASON	REASON_SUPPLY_INCR
SCINC.QUANTITY	WEIGHT
SCINDUMP.INTRVL	INTERVAL
SCINDUMP.TIME	TIME
SCINDUMP.SIDE	SIDE
SCINDUMP.TARGET	NAME_TARGET_SHORT
SCINDUMP.CATEGORY	NAME_SUPPLY_CATEGORY
SCINDUMP.QUANTITY	WEIGHT

ATTRIBUTE DOMAIN CORRESPONDENCE

ATTRIBUTE	DOMAIN
SCLOST.INTRVL	INTERVAL
SCLOST.TIME	TIME
SCLOST.SIDE	SIDE
SCLOST.UNIT	NAME_UNIT_SHORT
SCLOST.CATEGORY	NAME_SUPPLY_CATEGORY
SCLOST.REASON	REASON_SUPPLY_LOST
SCLOST.QUANTITY	WEIGHT
SCLOST.ACTION	SUPPLY_CATEGORY_LOST
SCONHAND.INTRVL	INTERVAL
SCONHAND.TIME	TIME
SCONHAND.SIDE	SIDE
SCONHAND.UNIT	NAME_UNIT_SHORT
SCONHAND.CATEGORY	NAME_SUPPLY_CATEGORY
SCONHAND.QUANTITY	WEIGHT
SCRECD.INTRVL	INTERVAL
SCRECD.TIME	TIME
SCRECD.SIDE	SIDE
SCRECD.UNIT	NAME_UNIT_SHORT
SCRECD.CATEGORY	NAME_SUPPLY_CATEGORY
SCRECD.REASON	REASON_SUPPLY_RECD
SCRECD.QUANTITY	WEIGHT
SCSENT.INTRVL	INTERVAL
SCSENT.TIME	TIME
SCSENT.SIDE	SIDE
SCSENT.UNIT	NAME_UNIT_SHORT
SCSENT.CATEGORY	NAME_SUPPLY_CATEGORY
SCSENT.REASON	REASON_SUPPLY_SENT
SCSENT.QUANTITY	WEIGHT
SCSHORT.INTRVL	INTERVAL
SCSHORT.TIME	TIME
SCSHORT.SIDE	SIDE
SCSHORT.UNIT	NAME_UNIT_SHORT

ATTRIBUTE DOMAIN CORRESPONDENCE

ATTRIBUTE	DOMAIN
SCSHORT.CATEGORY	NAME_SUPPLY_CATEGORY
SCSHORT.REASON	REASON_SUPPLY_SHORT
SCSHORT.QUANTITY	WEIGHT
SCTRANS.INTRVL	INTERVAL
SCTRANS.TIME	TIME
SCTRANS.SIDE	SIDE
SCTRANS.UNIT	NAME_UNIT_SHORT
SCTRANS.REASON	REASON_SUPPLY_TRANS
SCTRANS.DRY_WT	WEIGHT
SCTRANS.WET_WT	WEIGHT
SUPPLIES.CODE	CODE_SUPPLY_CATEGORY
SUPPLIES.SIDE	SIDE
SUPPLIES.CATEGORY	NAME_SUPPLY_CATEGORY
TALIVE.INTRVL	INTERVAL
TALIVE.TIME	TIME
TALIVE.SIDE	SIDE
TALIVE.UNIT	NAME_UNIT_SHORT
TALIVE.QUANTITY	QUANTITY
TARGETS.INTRVL	INTERVAL
TARGETS.TIME	TIME
TARGETS.ID	NAME_TARGET_SHORT
TARGETS.NAME	NAME_TARGET_LONG
TARGETS.CATEGORY	TARGET_CATEGORY
TAVAIL.INTRVL	INTERVAL
TAVAIL.TIME	TIME
TAVAIL.SIDE	SIDE
TAVAIL.UNIT	NAME_UNIT_SHORT
TAVAIL.QUANTITY	QUANTITY
TGADA.INTRVL	INTERVAL
TGADA.TIME	TIME
TGADA.ID	NAME_TARGET_SHORT
TGADA.STATUS	STATUS

ATTRIBUTE DOMAIN CORRESPONDENCE

ATTRIBUTE	DOMAIN
TGCAPABLE.INTRVL	INTERVAL
TGCAPABLE.TIME	TIME
TGCAPABLE.ID	NAME_TARGET_SHORT
TGCAPABLE.ACTION	TARGET_CAPABILITY
TGCAPABLE.REASON	REASON_TARGET_CAP
TGCAPABLE.PCT_CAPABLE	PERCENTAGE
TGDETECT.INTRVL	INTERVAL
TGDETECT.TIME	TIME
TGDETECT.ID	NAME_TARGET_SHORT
TGDETECT.SIDE	SIDES
TGDETECT.REASON	REASON_TARGET_DET
TGRANGE.INTRVL	INTERVAL
TGRANGE.TIME	TIME
TGRANGE.ID	NAME_TARGET_SHORT
TGRANGE.RNG	TARGET_RANGE
TGSIDE.INTRVL	INTERVAL
TGSIDE.TIME	TIME
TGSIDE.ID	NAME_TARGET_SHORT
TGSIDE.SIDE	SIDES
TGSIZE.INTRVL	INTERVAL
TGSIZE.TIME	TIME
TGSIZE.ID	NAME_TARGET_SHORT
TGSIZE.SIZE	TARGET_SIZE
TGUNIT.INTRVL	INTERVAL
TGUNIT.TIME	TIME
TGUNIT.ID	NAME_TARGET_SHORT
TGUNIT.UNIT	NAME_UNIT_SHORT
TGUNIT.LAT	LATITUDE
TGUNIT.LON	LONGITUDE
TRKILLED.INTRVL	INTERVAL
TRKILLED.TIME	TIME
TRKILLED.SIDE	SIDE

ATTRIBUTE DOMAIN CORRESPONDENCE

ATTRIBUTE	DOMAIN
TRKILLED.UNIT	NAME_UNIT_SHORT
TRKILLED.CARGOS	QUANTITY
TRKILLED.TANKERS	QUANTITY
TRKILLED.REASON	REASON_TRUCK_KILL
UNITS.SHORT_NAME	NAME_UNIT_SHORT
UNITS.LONG_NAME	NAME_UNIT_LONG
UNITS.TYPE	UNIT_TYPE
UNITS.SUBTYPE	UNIT_SUBTYPE
UNITS.SIDE	SIDE
UNITS.AIRCRAFT	QUANTITY
UTADA.INTRVL	INTERVAL
UTADA.TIME	TIME
UTADA.SIDE	SIDE
UTADA.UNIT	NAME_UNIT_SHORT
UTADA.STATUS	STATUS
UTAIRBASE.INTRVL	INTERVAL
UTAIRBASE.TIME	TIME
UTAIRBASE.SIDE	SIDE
UTAIRBASE.UNIT	NAME_UNIT_SHORT
UTAIRBASE.AIRBASE	NAME_UNIT_SHORT
UTAIRBASE.REASON	REASON_AIRBASE
UTARRIVES.INTRVL	INTERVAL
UTARRIVES.TIME	TIME
UTARRIVES.SIDE	SIDE
UTARRIVES.UNIT	NAME_UNIT_SHORT
UTARRIVES.LAT	LATITUDE
UTARRIVES.LON	LONGITUDE
UTCAS.INTRVL	INTERVAL
UTCAS.TIME	TIME
UTCAS.SIDE	SIDE
UTCAS.UNIT	NAME_UNIT_SHORT
UTCAS.SQUADRON	NAME_UNIT_SHORT

ATTRIBUTE DOMAIN CORRESPONDENCE

ATTRIBUTE	DOMAIN
UTCAS.NO_AIRCRAFT	QUANTITY
UTCONTACT.INTRVL	INTERVAL
UTCONTACT.TIME	TIME
UTCONTACT.UNIT1	NAME_UNIT_SHORT
UTCONTACT.UNIT2	NAME_UNIT_SHORT
UTCONTACT.STATUS	STATUS
UTCONTACT.POSTURE1	UNIT_POSTURE
UTCONTACT.POSTURE2	UNIT_POSTURE
UTDELAYED.INTRVL	INTERVAL
UTDELAYED.TIME	TIME
UTDELAYED.SIDE	SIDE
UTDELAYED.UNIT	NAME_UNIT_SHORT
UTDELAYED.DELAYER_SIDE	SIDES
UTDELAYED.LAT	LATITUDE
UTDELAYED.LON	LONGITUDE
UTDELAYED.DURATION	TIME
UTHQ.INTRVL	INTERVAL
UTHQ.TIME	TIME
UTHQ.SIDE	SIDE
UTHQ.UNIT	NAME_UNIT_SHORT
UTHQ.HQ	NAME_UNIT_SHORT
UTHQ.REASON	REASON_HQ
UTINCAR.INTRVL	INTERVAL
UTINCAR.TIME	TIME
UTINCAR.SIDE	SIDE
UTINCAR.UNIT	NAME_UNIT_SHORT
UTINCAR.INC	NAME_UNIT_SHORT
UTLIFTED.INTRVL	INTERVAL
UTLIFTED.TIME	TIME
UTLIFTED.SIDE	SIDE
UTLIFTED.UNIT	NAME_UNIT_SHORT
UTLIFTED.LIFTER	NAME_UNIT_SHORT

ATTRIBUTE DOMAIN CORRESPONDENCE

ATTRIBUTE	DOMAIN
UTLIFTED.STATUS	STATUS
UTLIFTED.REASON	REASON_UNIT_LIFT
UTPOSTURE.INTRVL	INTERVAL
UTPOSTURE.TIME	TIME
UTPOSTURE.SIDE	SIDE
UTPOSTURE.UNIT	NAME_UNIT_SHORT
UTPOSTURE.NEW_POSTURE	POSTURE
UTPOSTURE.OLD_POSTURE	POSTURE
UTPOSTURE.REASON	REASON_UNIT_POSTURE
UTPOSTURE.LAT	LATITUDE
UTPOSTURE.LON	LONGITUDE
UTREINF.INTRVL	INTERVAL
UTREINF.TIME	TIME
UTREINF.SIDE	SIDE
UTREINF.UNIT	NAME_UNIT_SHORT
UTREINF.REINFORCER	NAME_UNIT_SHORT
UTREINF.STATUS	STATUS
UTSTART.INTRVL	INTERVAL
UTSTART.TIME	TIME
UTSTART.SIDE	SIDE
UTSTART.UNIT	NAME_UNIT_SHORT
UTSTART.LAT	LATITUDE
UTSTART.LON	LONGITUDE
UTSTART.DEST_LAT	LATITUDE
UTSTART.DEST_LON	LONGITUDE
UTSTOP.INTRVL	INTERVAL
UTSTOP.TIME	TIME
UTSTOP.SIDE	SIDE
UTSTOP.UNIT	NAME_UNIT_SHORT
UTSTOP.LAT	LATITUDE
UTSTOP.LON	LONGITUDE
UTSTOP.REASON	REASON_UNIT_STOP

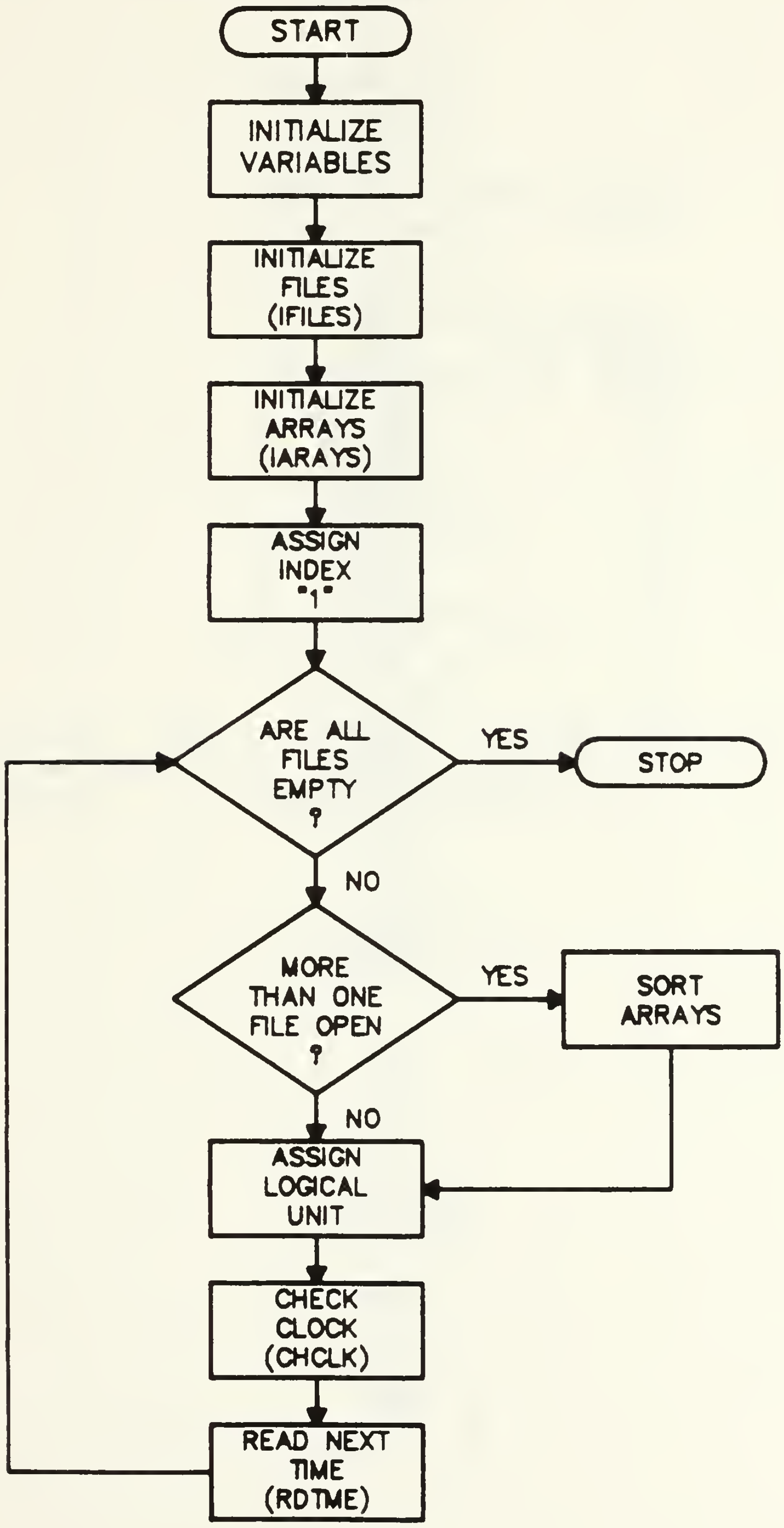
ATTRIBUTE DOMAIN CORRESPONDENCE

ATTRIBUTE	DOMAIN
UTSTRENGTH.INTRVL	INTERVAL
UTSTRENGTH.TIME	TIME
UTSTRENGTH.SIDE	SIDE
UTSTRENGTH.UNIT	NAME_UNIT_SHORT
UTSTRENGTH.STRENGTH	PERCENTAGE
UTSUPPORT.INTRVL	INTERVAL
UTSUPPORT.TIME	TIME
UTSUPPORT.SIDE	SIDE
UTSUPPORT.UNIT	NAME_UNIT_SHORT
UTSUPPORT.SUPPORT_UNIT	NAME_UNIT_SHORT
UTSUPPORT.REASON	REASON_UNIT_SUPPORT
UTTRANS.INTRVL	INTERVAL
UTTRANS.TIME	TIME
UTTRANS.SIDE	SIDE
UTTRANS.UNIT	NAME_UNIT_SHORT
UTTRANS.DRY_WT	WEIGHT
UTTRANS.WET_WT	WEIGHT
UTTRANS.REASON	REASON_UNIT_TRANS
WEAPONS.CODE	CODE_AIR_WEAPON
WEAPONS.TYPE	NAME_AIR_WEAPON
WEAPONS.SIDE	SIDE
WPNEXPEND.INTRVL	INTERVAL
WPNEXPEND.TIME	TIME
WPNEXPEND.SIDE	SIDE
WPNEXPEND.UNIT	NAME_UNIT_SHORT
WPNEXPEND.MISSION	CODE_AIR_MISSION
WPNEXPEND.QUANTITY	QUANTITY
WPNEXPEND.TYPE	CODE_AIR_WEAPON
WPNEXPED.REASON	REASON_EXPEND

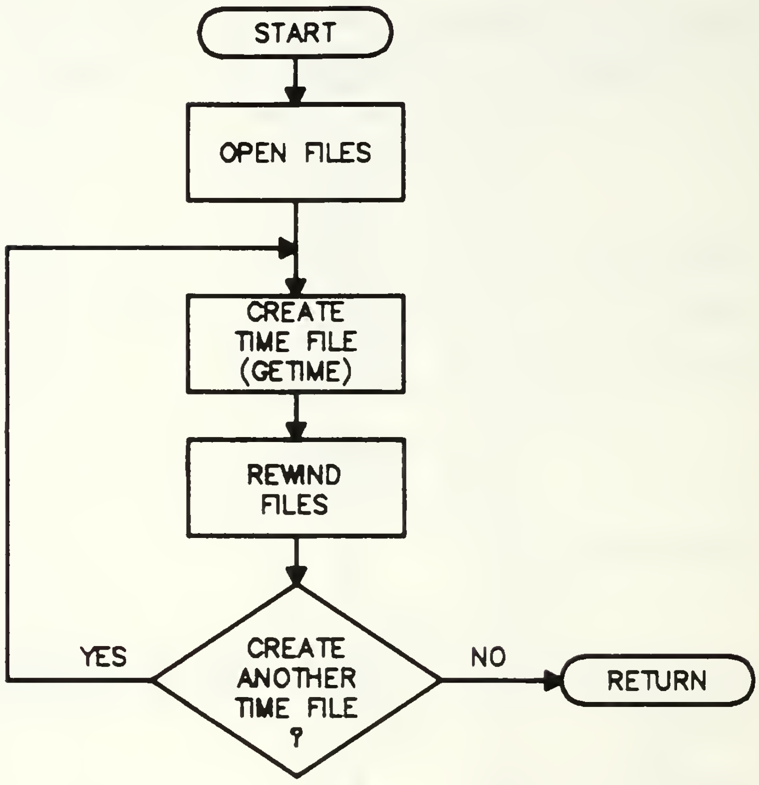
APPENDIX I

FLOW CHARTS

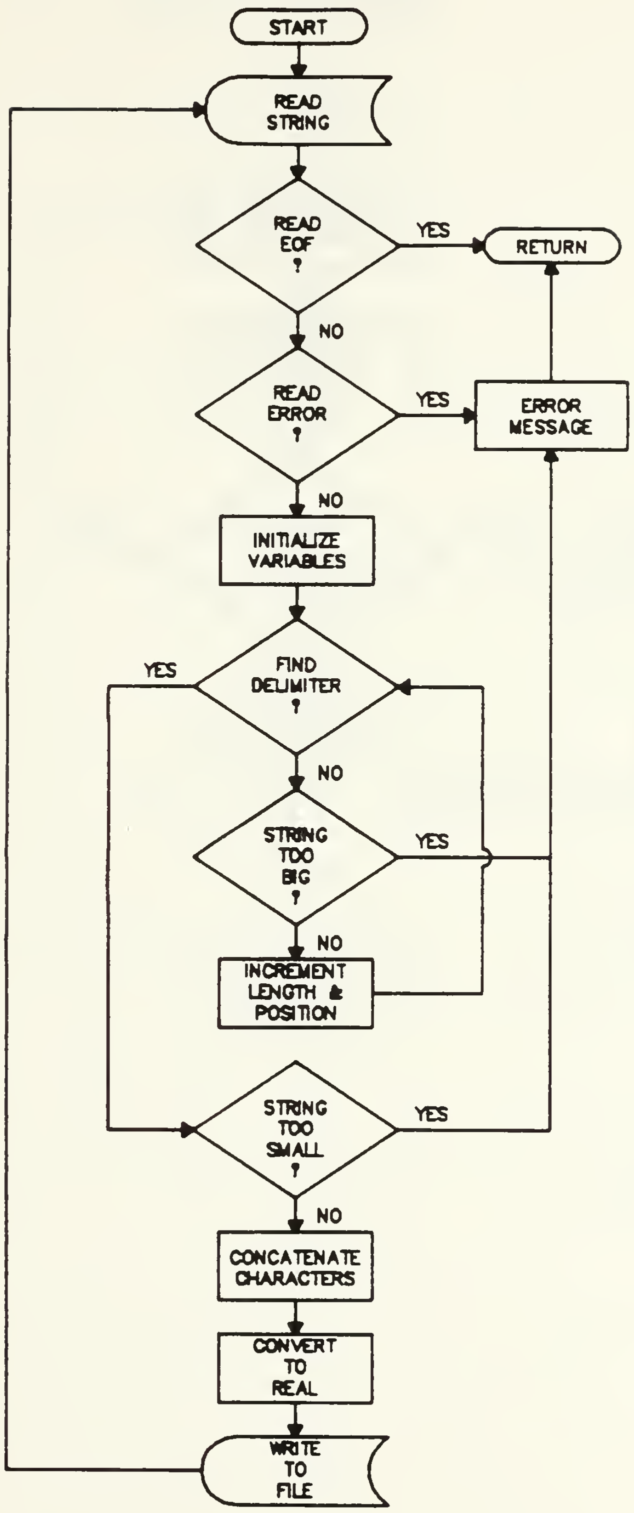
Program PUSH MAIN



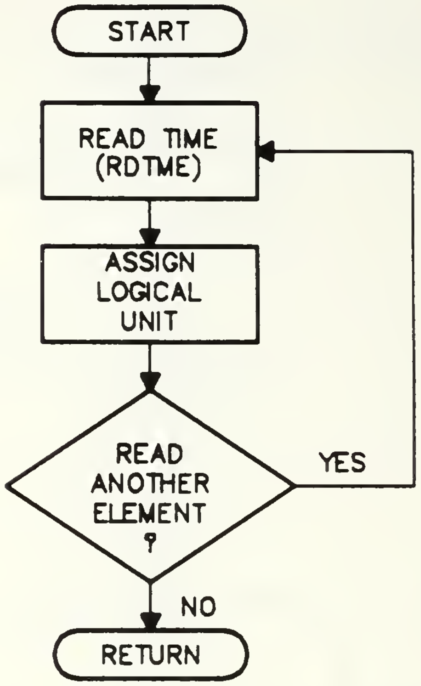
Subroutine IFILES



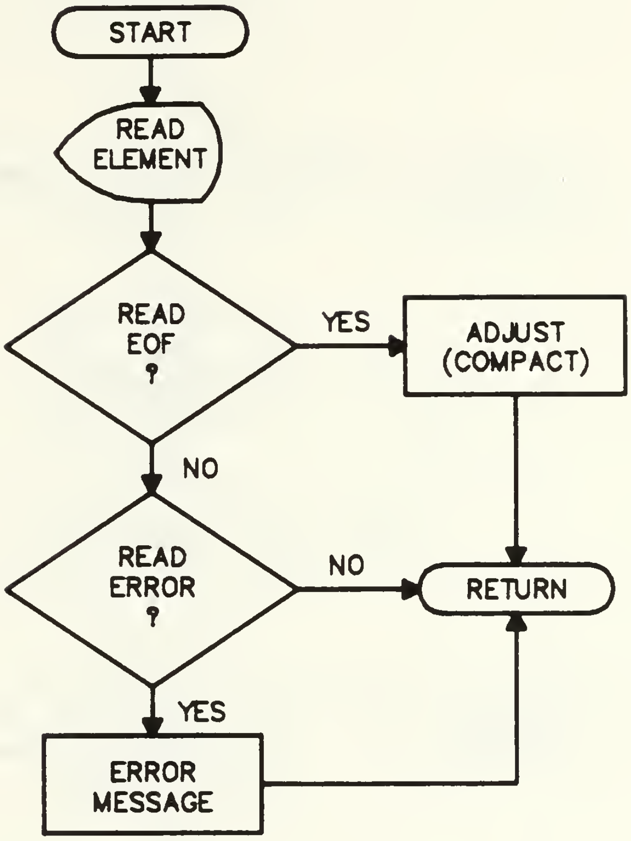
Subroutine GETIME



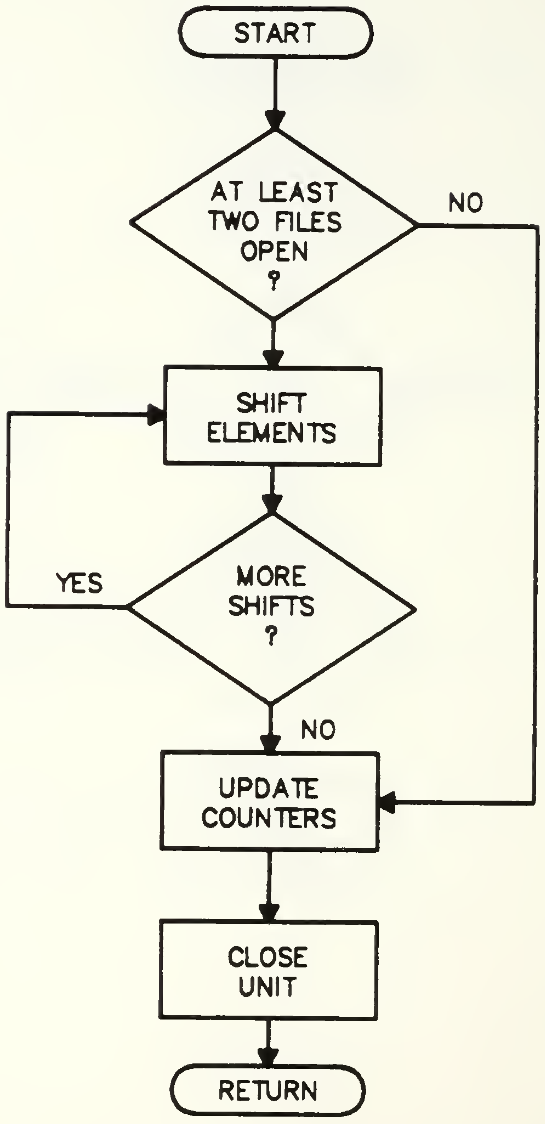
Subroutine IARAYS



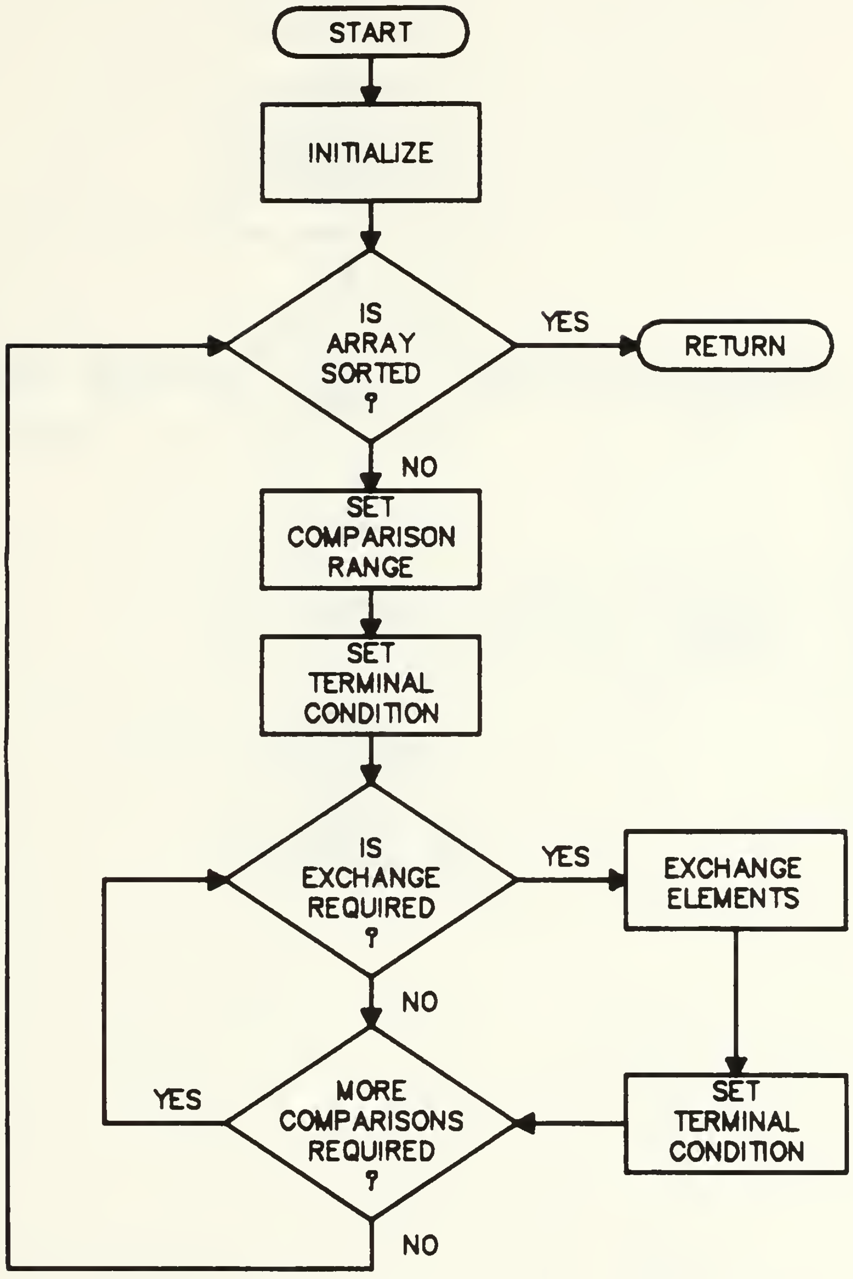
Subroutine RDTME



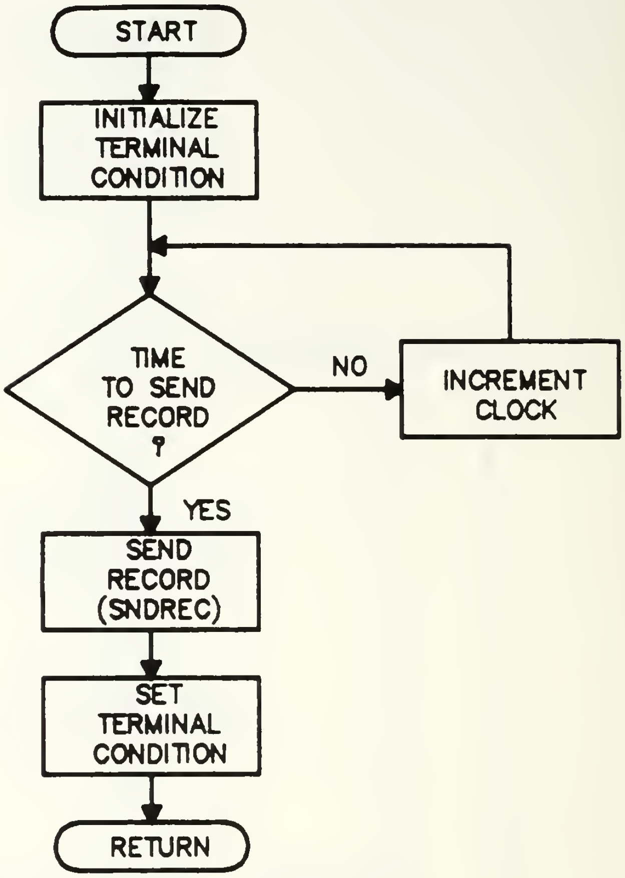
Subroutine COMPACT



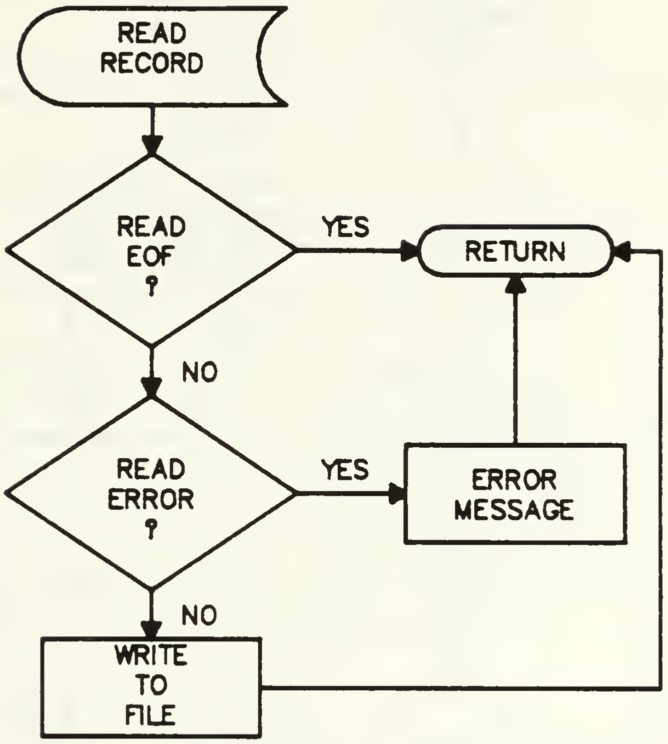
Subroutine SORT



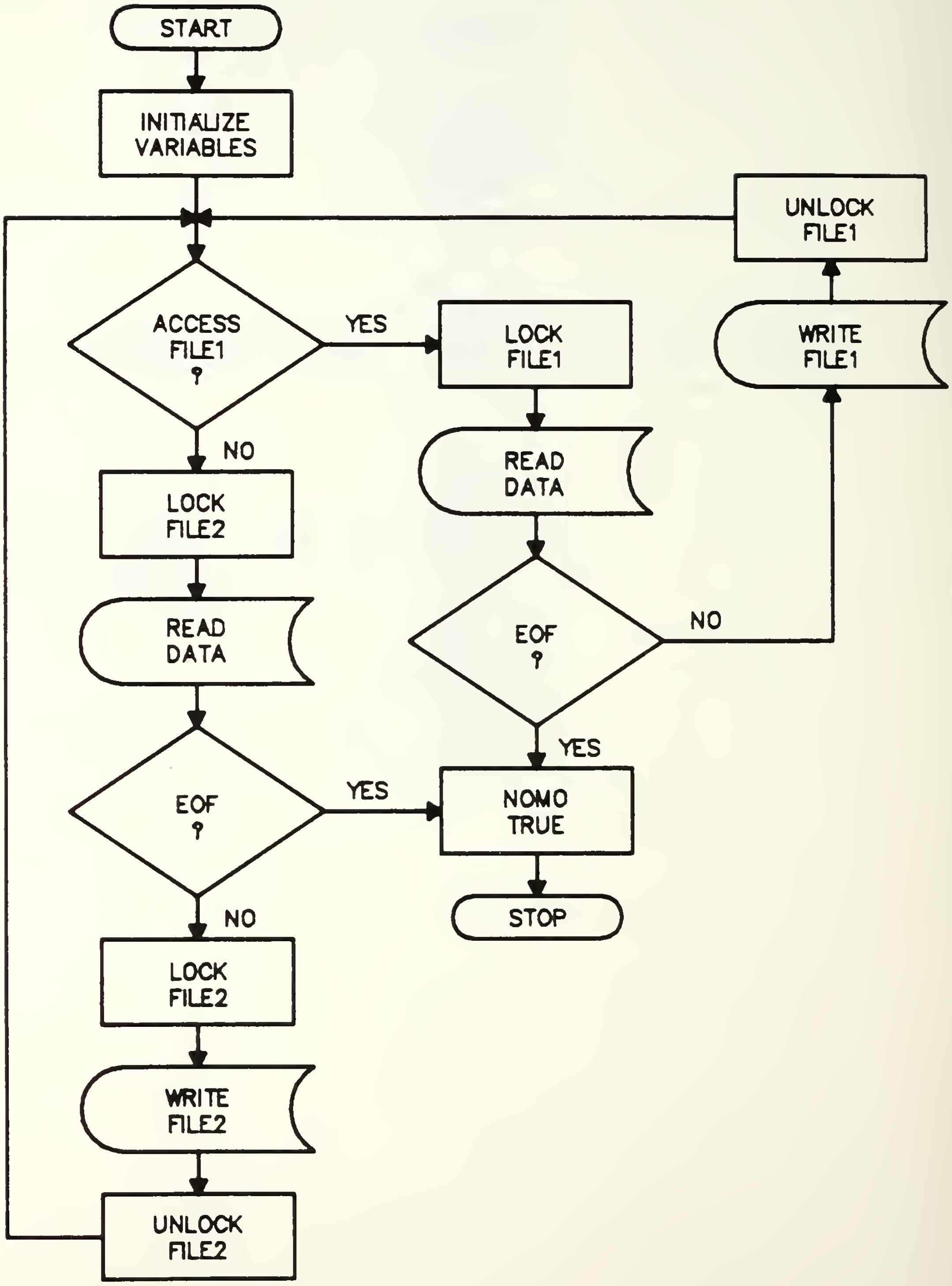
Subroutine CHCLK



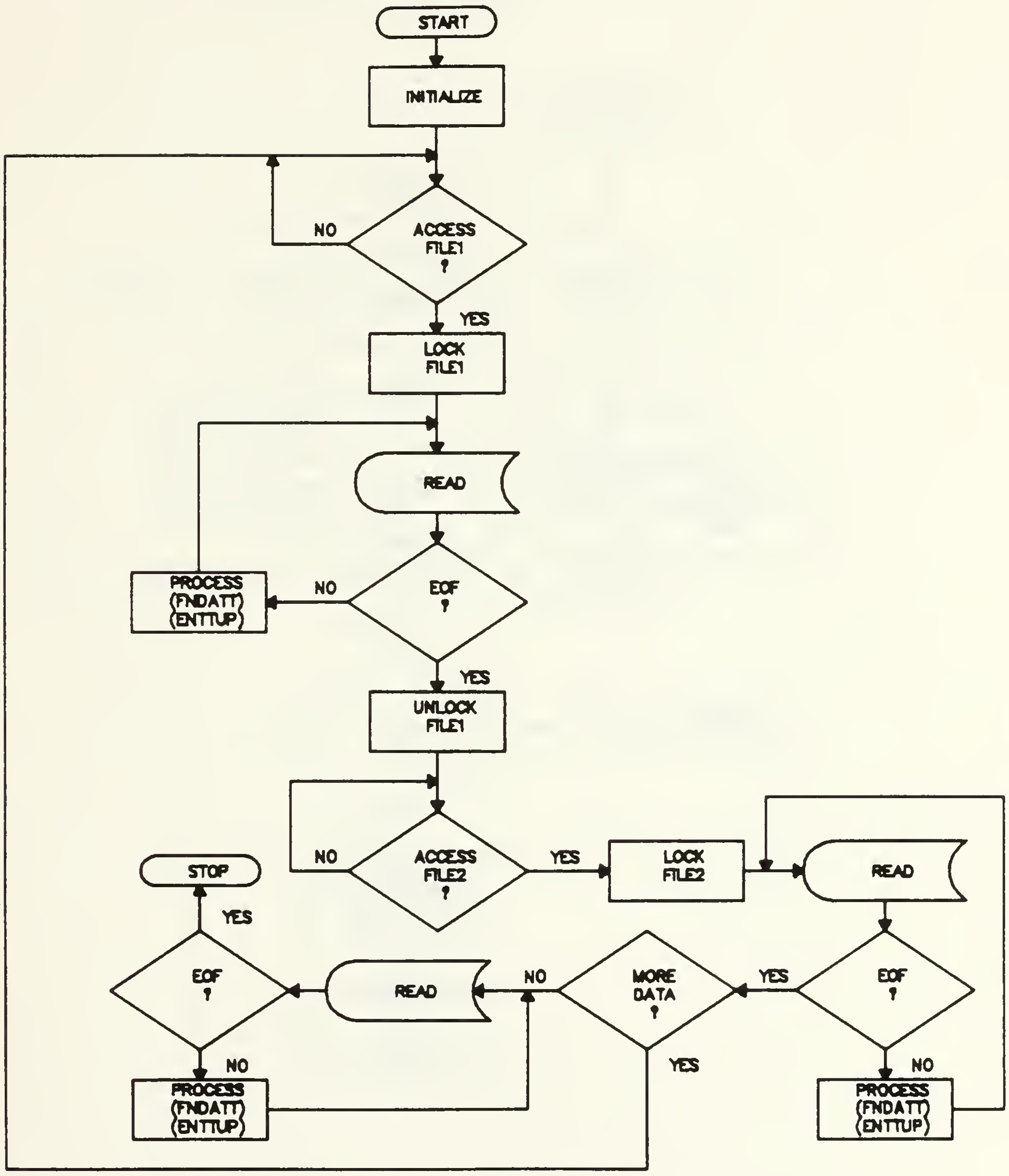
Subroutine SNDREC



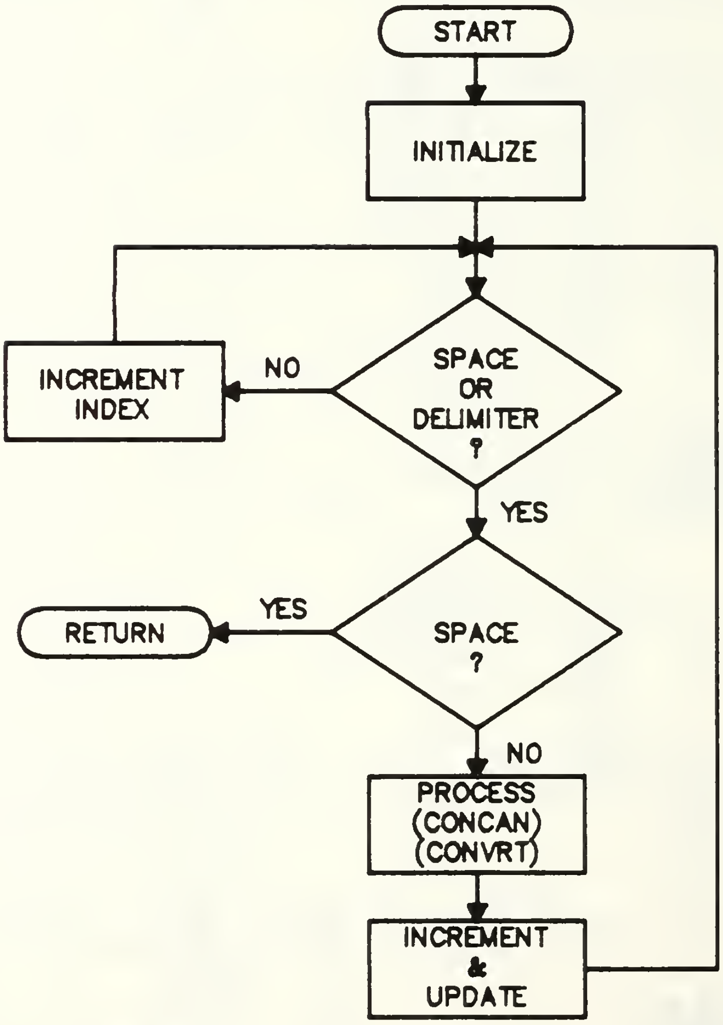
Program PULL



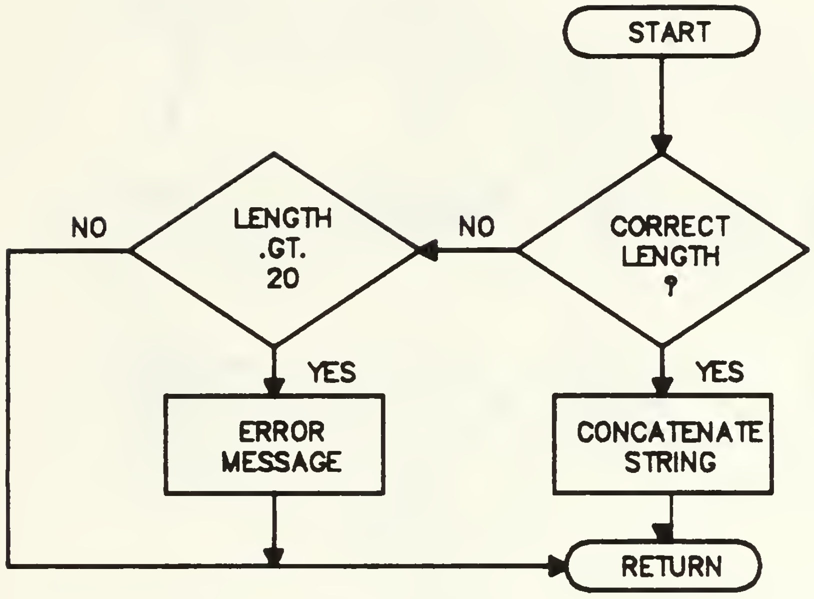
Program PACK



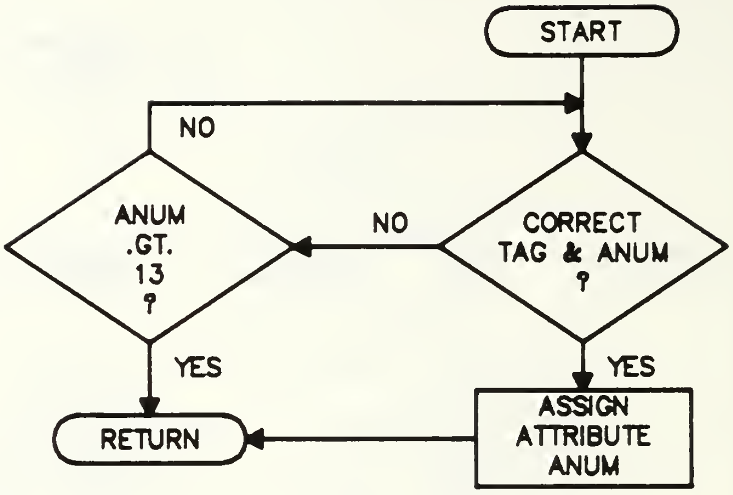
Subroutine FNDATT



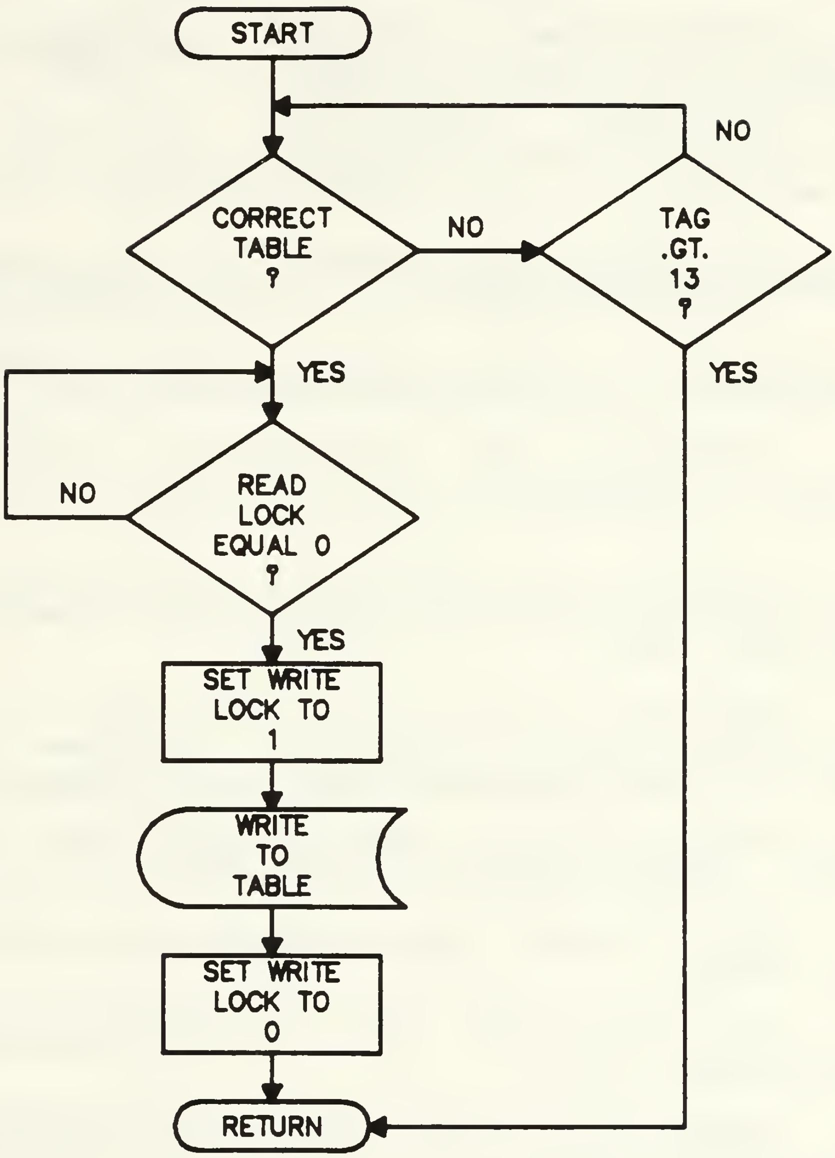
Subroutine CONCAN



Subroutine CONVRT



Subroutine ENTTUP



LIST OF REFERENCES

1. Conwell, Candance Lee, Unique Considerations in the Design of a Command and Control Decision System, M.S. Thesis, Naval Postgraduate School, Monterey, CA, June 1983.

2. Department of the Army, TRADOC Pam 11-8, Studies and Analysis Handbook, Fort Monroe, VA, July 1985.

3. Carroll, Michael F., A Data Base Design for a Multimedia C2 Workstation in Support of RESA, M.S. Thesis, Naval Postgraduate School, Monterey, CA, March 1987.

4. Syscon Corporation, Postprocessor User Guide, Washington, D. C., May 1987.

5. Jet Propulsion Laboratiry, JTLS Executive Overview, Pasadena, CA., May 1986.

6. Interview between Ellen Rolands, Rolands and Assoc., Monterey, CA., and the author, February 1988.

7. Svobodova, Liba, "Performance Problems in Distributed Systems," Information, Volume 18, Number 1, February 1980, pp. 21 - 38.

8. Stonebraker, M., "The Design and Implementation of INGRES", ERL Technical Memorandum Reprint 1468, University of California, September, 1976.

9. Chamberlin, D. D., J. N. Gray and I. L. Traiger, "Views, Authorization and Locking in a Relational Data BaseSystem", Proc. 1975 AFIPS National Computer Conference, 1975.

10. Ellison, Lawrence, ORACLE Overview and Introduction to SQL, 2nd ed., ORACLE Corporation, May 1985.

11. Comer, Douglas, Internetworking with TCP/IP, Prentice-Hall, Inc., 1988.

12. Sun Microsystems, Mail and Messages Beginners Guide, Sun Microsystems, 1986.

13. SYSCON Corporation, Modern Aids to Planning Program, Data Requirements Manual, JTLS Version 1.6", Washington, D.C., August 1987.

14. Kroenke, David M., Data Base Processing, 2nd ed., Chicago: Science Research Associates, Inc., 1983.

15. ORACLE Corporation Incorporated, ORACLE for Sun Workstation and User's Guide, Belmont, CA., July 1986.

16. Hammer, M. and McLeod, D., "Data Base Description with SDM: A Semantic Data Base Model," ACM Transactions on Data Base Systems, Volume 6, Number 3, September 1981, pp. 351 - 386.

17. Bernstein, Philip and Goodman, Nathan, "Concurrency Control in Distributed Database Systems", Computer Surveys, Vol. 13, No. 2, Jun 1981.

18. SYSCON Corporation, JTLS User Guide, Version 1.6, Washington, D.C., August 1987.

19. ORACLE Corporation Incorporated, ORACLE For Sun Workstation/Sun UNIX System Release Bulletin, Belmont, CA., August 1987.

20. Naval Oceans Systems Center, Report NOSC TR 1006, Menu-based Natural Language Query for Naval Command and Control, by G. A. Osga, December 1984.

21. Sun Microsystms, Sun Systems Overview, Sun Microsystems, 1986.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	CDR J. Stewart, Code 55ST Naval Postgraduate School Monterey, CA 993943-5000	2
4.	Superintendent, Code 39 Attn: CPT. Charles Dunn III Naval Postgraduate School Monterey, CA 993943-5000	2
5.	CPT. Stanley H. Evans Jr. 446 Wellesley Rd. Philadelphia, PA 19119	2

[illegible]3 - 899